ETS – ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMPARISON BETWEEN ANSYS AND ABAQUS USING ULTRASONIC GUIDED WAVES.

by

Lluís Ortiz Alcalá

Department of Mechanical Engineering

École de Téchnologie Supérieure

Montréal, Canada.

July 2014

Abstract

The detection of corrosion in pipelines is an aspect of utmost importance as far as the fluid transportation industry is concerned, and especially if they have a difficult access. As they are installed outdoors or underground, these kilometric conduits are under various weather and natural phenomena, which may damage them causing loses and consequently serious natural disasters.

Over the years, the most common way to check the pipelines' thickness has been thanks to ultrasonic and Foucault currents. The constraint of these techniques is that they only allow examining the thickness just under the probe. Thus, the checking procedure along its entire length becomes a very dull and long process. The problem arises when the access to certain stretches of those pipelines becomes extremely complex, making this control impossible. Upon needing to inspect these pipelines in a precise, quickly and easy way along their length, this control method is rarely used any more.

Low frequency guided waves are beginning to be used frequently to perform this task since their detection and location capability of corroded patches is very good. The only drawback that this screening technique presents is that it only gives a rough estimate of the remaining wall thickness. Currently, guided waves are the most powerful method not only to detect inaccessible corroded patches but also to map their remaining thickness precisely.

Nowadays, the world of computer science is so immensely vast that there is a large amount of software solutions to perform the same task. The problem ascends when a user needs to complete a specific job and does not know which of the programs at his disposal to choose. Some of the aspects that will prevail in his decision is if the software's interface is good-looking, the price of software's license, the memory consumption at the time of performing a task but mainly the time taken to complete the work.

The aim of this report is to compare two of the most important programs for finite elements simulations: Abaqus and ANSYS. This analysis was done by applying a signal of 50 kHz and 10 cycles, involved in a Hanning window in a titanium plate of 1 m long and with a thickness of 5 mm. The reason why this study was done by simulating a plate instead of a pipeline, without forgetting the aforementioned, is because it was needed to simulate an easy geometry in order to shorten the time taken to solve the simulation by both software solutions.

The conclusions of this work were obtained by analysing which of the software solutions offered the most accurate displacements of a node located one meter far from where the signal was implemented. In case that both were equally precise, what was determinant for us at the time of choosing the best software program was which of both took the highest analysis speed, less time and memory consumption to perform the task.

Acknowledgements

This Final Degree Project would not have been possible without the help offered by a number of people who have strongly contributed to my personal and professional development as well as my weal during this period in Montréal.

At fist, my deepest thanks to Prof. Pierre Belanger for his guidance, encouragement and help offered during this time as tutor of this project. If it were not for Prof. Pierre Belanger theoretical knowledge about Lamb Waves as well as the tools used in this project, this thesis could not have been developed.

Special thanks to the ETS professors of *Introduction à l'ingénierie de la réadaptation,* Prof. Rachid Aissaoui and Prof. Pierre-Olivier Lemieux for learning me both the basis of Matlab, a software which has been essential for the project development.

For his constant encouragement, happiness and unyielding support, I would like to thank my father, Luis and my mother, Natividad. Thanks for making this experience possible.

Contents

Chapter 1

Introduction

1.1 Motivation

ANSYS is an engineering simulation software based on the finite element analysis whose applications are so vast that it proves to be very useful for lots of engineering disciplines. During my years in university I worked a lot with ANSYS but focussed on structural problems and material resistance questions. During those years, I discovered that one of its main drawbacks was the inability to undo the work done in case of error and / or change in the modeling of any geometry. This was the main reason that pushed me to do some research in how to use the Matlab software for coding the type of simulation that wanted to study: the propagation of a Lamb wave through a titanium plate. The reason to chose this work field was to learn another important employment of this powerful program while helping its users to chose between 2 similar computer software solutions.
As mentioned before, a variable script was looked for so that it could be used for any geometry and any signal. That is why, the vast majority of the simulation parameters coded with Matlab were replaced with variables; otherwise that Matlab code would have been useful just for a unique plate profile.

1.2 Project outline

This project has been divided in the topics described here below.

At first, a review based on guided waves as well as their applications and the simulation of guided waves will be given in chapter 2.

Properly, this study begins by learning about how to use ANSYS so as to simulate guided waves. Before coding any simulation phase, the visual interface was used in order to get familiar with the software and know the necessary stages that the simulation required. In Chapter 3, all the stages of the first simulation of our plate are described. This initial simulation process describes from how to create the plate

section to how to obtain and compare the results obtained, through how to mesh the geometry, set the material properties, create new analysis and state both the initial and the boundary conditions.
As the guided waves were simulated with this sort of programs for the first time, some errors appeared; those mistakes were identified by comparing the results obtained after the simulation were with the reference ones. Once those faults were corrected the coding procedure could get started.

Chapter 4 describes in detail not only how the script was done but also how each phase of the simulation was coded. In this fragment of the report some important steps such as how to create and mesh a geometry, code a signal function, set a new analysis or state the signal regimes is explained. In addition, it is explained how the script generated the input file necessary for ANSYS in order to build the plate and start its simulation. Finally this chapter explains how the results were compare with the purpose of proving if they were correct.

In the last chapter, the conclusions of the project are exposed and explained. Not only the numerical results are shown but also the choice made for determining which of the software solutions is better.

The input ANSYS file as well as the complete Matlab script would be included in an appendix at the end of the document.

Chapter 2

Guided Waves' review.

Before analyzing the different displacements of the plate as result of the implementation of the signal at one of its nodes, it is convenient to study how guided waves propagate and how are their interactions with defects. Before talking about the different types, it is suitable to talk about guided waves themselves. A guided wave is a wave that is guided by the boundaries of the structure in which they propagate. There are different types of guided waves such as Rayleigh or Lamb waves inter alia. While the first kind of waves are free waves on a surface of a semi-infinite solid, Lamb waves are plane-strain solutions to a free plate problem; this last type of guided waves is the one this report will be focused on. Broadly, the main difference between Rayleigh in a free surface and Lamb waves in a plate is that in the second type there is a finite scale, such as the thickness of the plate. This means that for finite values of the ratio of the Lamb wavelength λ to the thickness, the Lamb waves have dispersive behaviour. Determination of the dispersion relation and hence the variation of phase (Vph) and group (VG) velocities with frequency is an important part of the problem.

As this background does not pretend to deepen in the mathematical background, just the lowest modes are going to be treated, the fundamental symmetric (S0) and antisymmetric (A0) modes. This fact leads us to the partial wave analysis, which is based on considering separately the two fundamental components of it, the longitudinal and transversal waves. Easily, the A0 mode displacements would look similar to a flag while the S0 mode displacements would resemble a very thin and long worm.

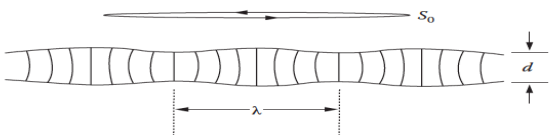


Figure 2.1: Symmetric fundamental mode. S0.

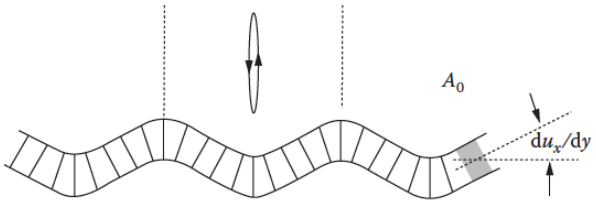


Figure 2.2: Antisymmetric mode. A0

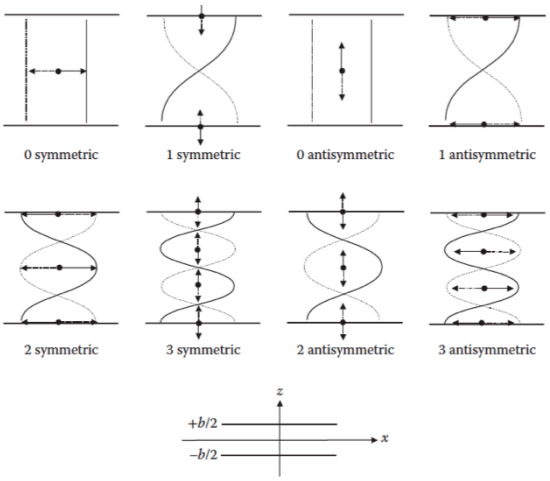


Figure 2.3: Wave shapes according to the different modes

As the main object of study was the y-axis displacement of the plate's nodes, it was needed to work with a very low frequency in order to avoid having different modes acting at the same time; if that had been the case, the displacement analysis would have been very hard to be performed. Everything was done with the purpose of resting always on the A0 mode, which was the most interesting for developing our study.

As mentioned, it is important to distinguish between the phase and the group velocity. The phase velocity represents the velocity at which a mode at a given frequency is traveling in a medium (for example that of a wave crest), while the group velocity is the velocity of propagation of a wave packet. In order to get familiarized with all this understanding, the program DISPERSE was used.

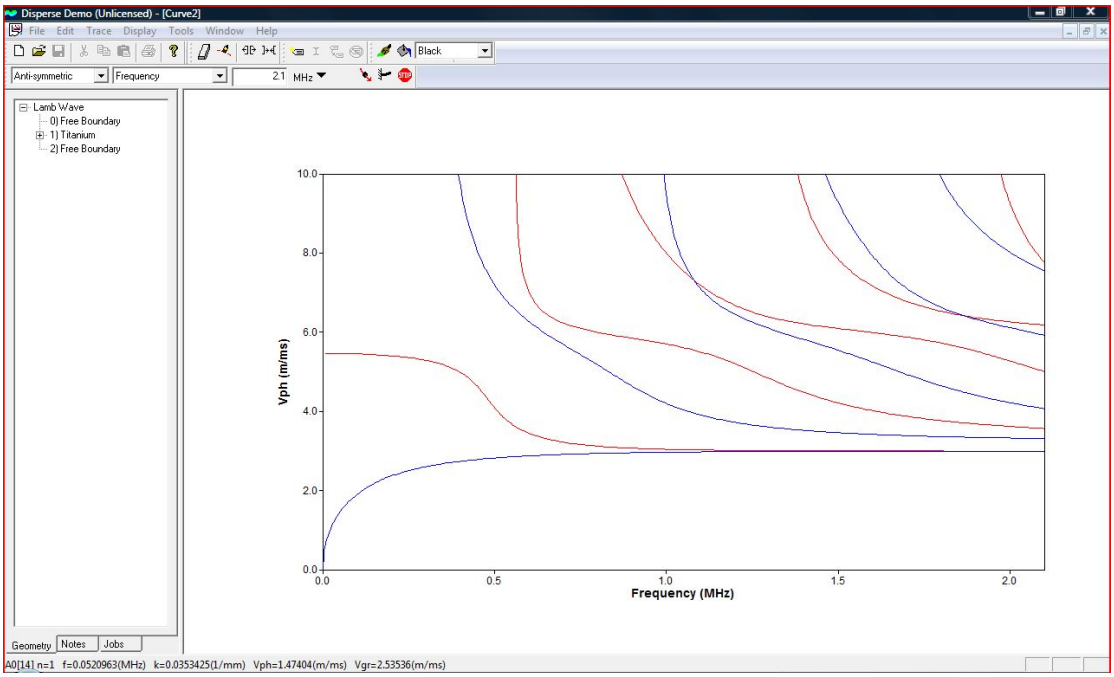


Figure 2.4: Phase Velocity graph in DISPERSE

Chapter 3

The first simulation

3.1 First settings

Though it does not seem like it, in **ANSYS 14.0 MECHANICAL APDL PRODUCT LAUCHER** is vital to choose the right software's license since each of them has limited functions. If a wrong choice was made, this fact could lead to the impossibility to perform the required simulation. Given this project, it was necessary for us the license "**Ansys Mechanical/Emag"** given that it allowed us to develop simulations for structural problems involving not only ultrasound but also electromagnetic waves.
Once the license necessary to solve our problem was chosen, we had to enlarge the number of substeps that the software could write in the results file after doing the simulation. By default, ANSYS writes a maximum of 1000 substeps in this kind of files, an amount that was not enough to treat lamb waves simulations accurately. So what was necessary was to write in the software command input bar the instruction "**/CONFIG,NRES,5000"**, a tool that extended the number of substeps from 1000 to 5000, an amount presumably large enough to complete our simulation rightly.

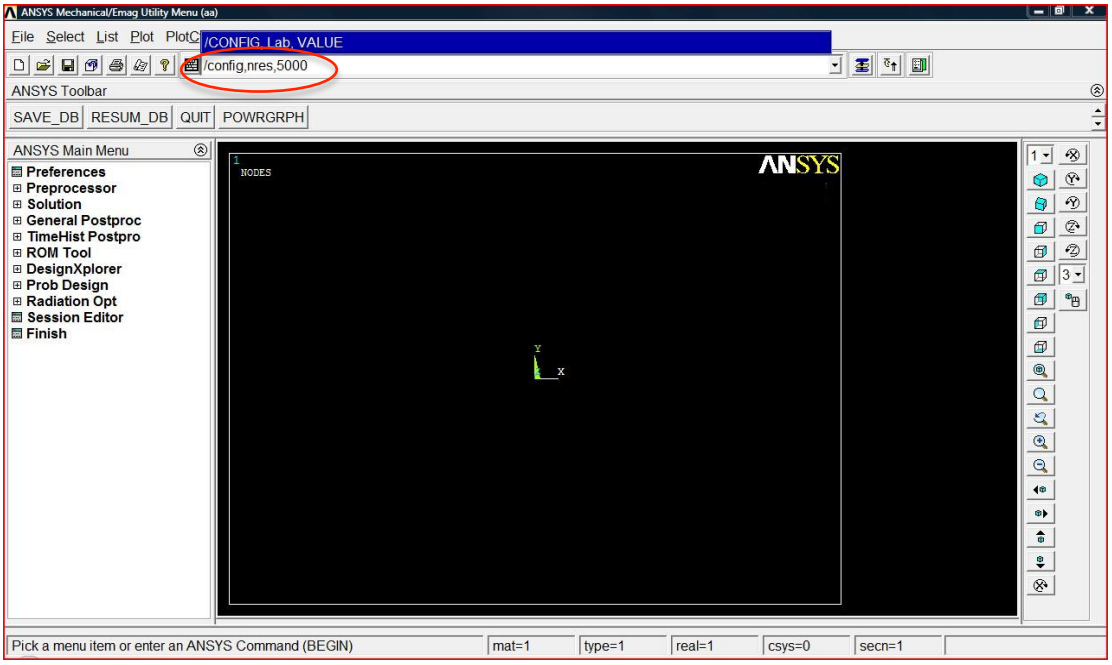


Figure 3.1.1: Instruction to enlarge the number of substeps

Another drawback of this finite elements software solution is that it is a computer program that does not indicate the units of the physical magnitudes the user is working with. With the purpose of introducing the input data correctly, it was needed to employ the international units system, which was fulfilled by introducing the instruction "**/UNITS,SI"** in the command input bar as well.

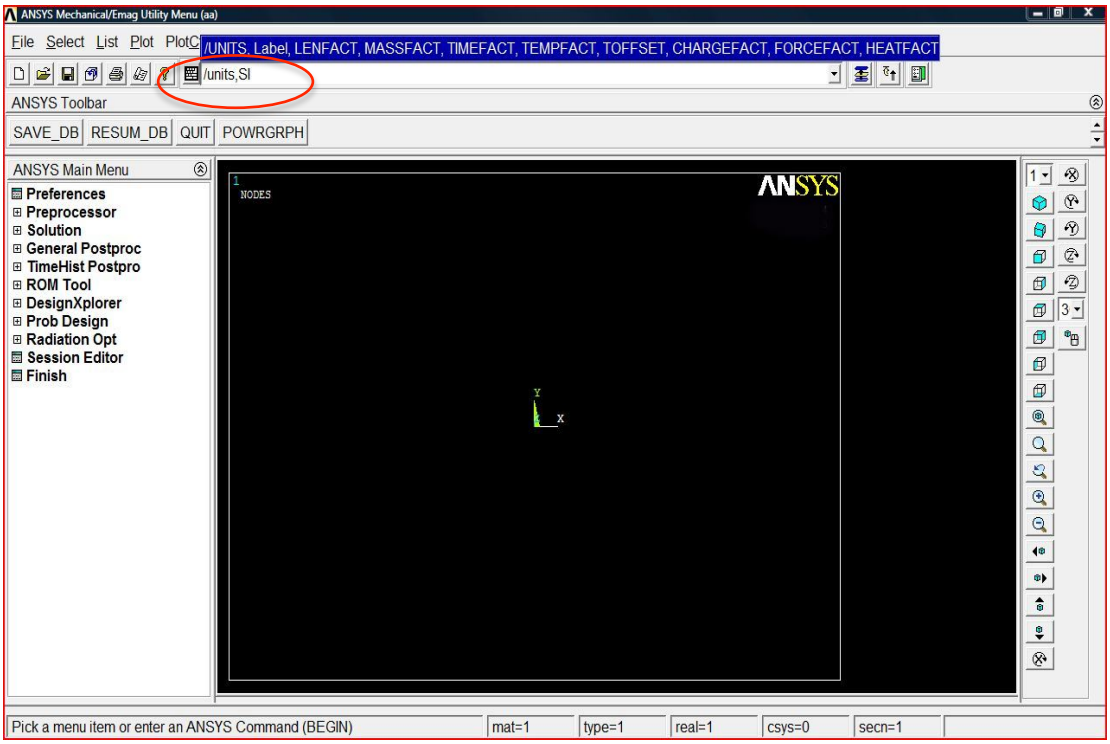


Figure 3.1.2: Instruction for working in the International Units System

The next phase for the modelling was choosing which of the analysis types our simulation required. In our case it needed to be structural for the reason that the main results that were of our interest were the plate displacements both in the y-axis and x-axis directions instead of thermal results or fluid magnitudes. The way this setting was applied was by following the procedure below.

Main Menu ➔ Preference ➔ Structural

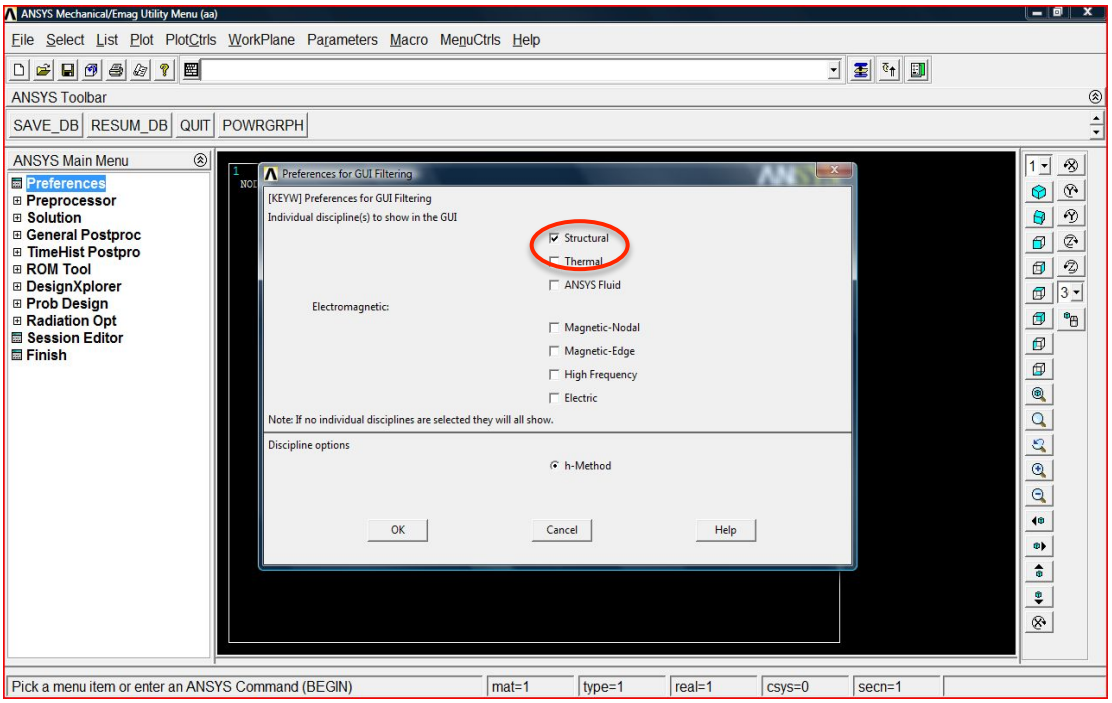


Figure 3.1.3: Selection of the structural Analysis

3.2 Modelling procedure

Reached this point, the next step consisted in creating the geometry of the plate. The easiest and quickest way to execute it was by creating a rectangle with the plate's section dimensions. To achieve it, these instructions needed to be done.

Main Menu ➔ Preprocessor ➔ Modelling ➔Create ➔ Areas ➔ Rectangle ➔ By 2 Corners.

A new window appeared in order to state the plate's section size by filling in the following blanks:

WP X / WP Y: The origin of the working plane in the directions X and Y (X=0.0;Y=0.0)
Width: Length of the plate (**1 m**)
Height: Thickness of the plate (**0.005 m**).

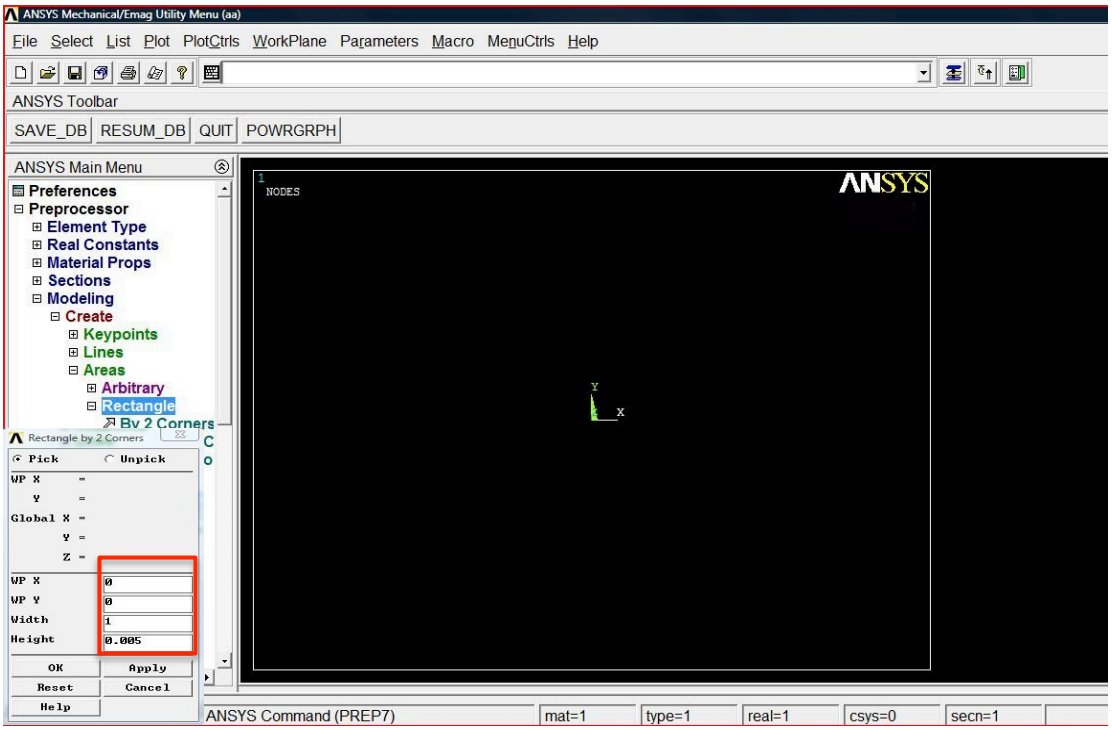


Figure 3.2.1: Plate's section dimensioning

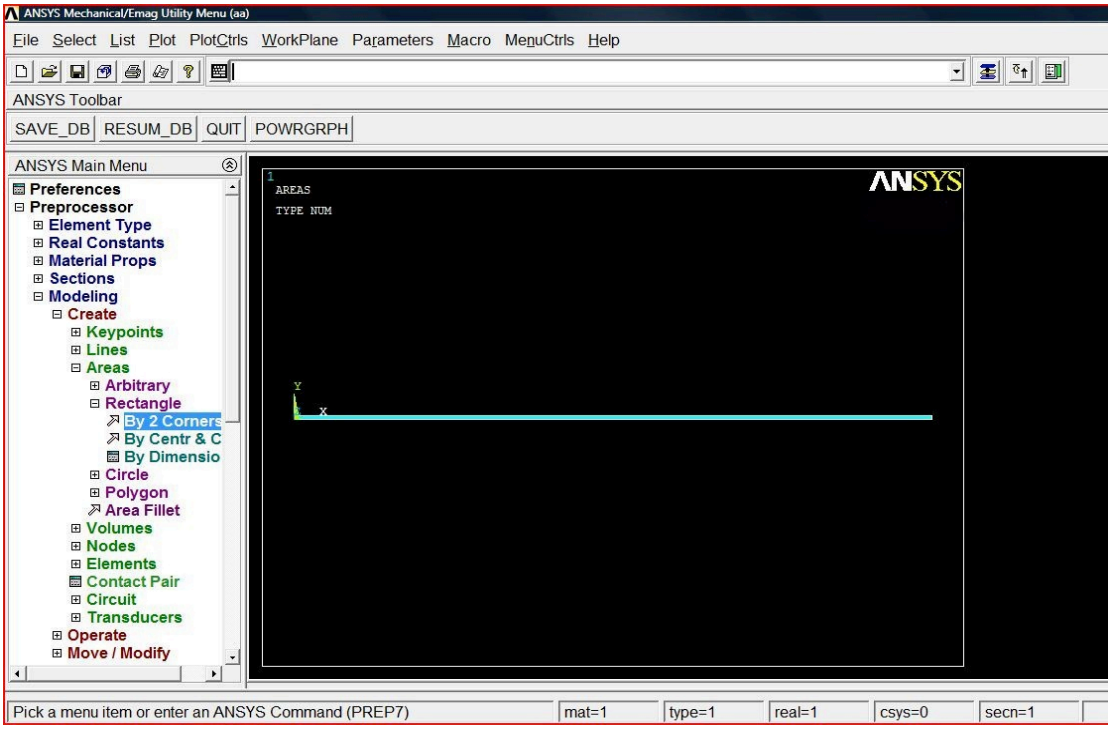


Figure 3.2.2: Plate's Section geometry

3.3 Element type and Material Properties

Thinking of the rectangular form of its section, it was thought that the best way to discretize the plate was dividing it into square elements instead of meshing it with octagonal elements, triangular ones or rather doing a free mesh. Moreover all the elements needed to be solid, as we were about to simulate a titanium plate. Thus, the most interesting element type for the simulation was the **"Quad 4 node 182"** not only because its input data included 4 nodes but also because the solution output related with the element was represented in nodal displacements in the overall nodal solution. The way to choose that element type was by realizing the approach below.

Main Menu ➔ Preprocessor ➔ Element Type ➔ Add/Edit/Delete ➔ Add

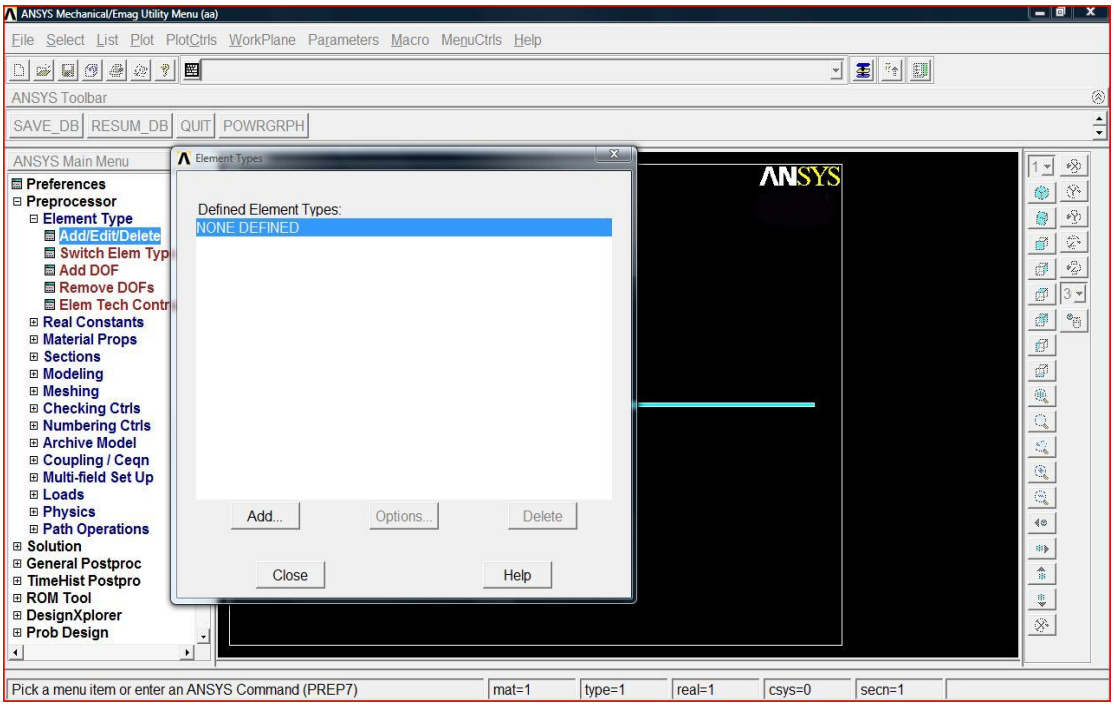


Figure 3.3.1: Element type definition procedure

Once more, a new window appeared in which the element type was selected. From the left list we had to choose **Structural**, corresponding to the analysis firstly defined. Next, from the emergent right list the "**Quad 4 node 182"** was chosen. As the project required working on the plate section, the "**Element Behaviour K3"** on the **"Options"** window needed to be changed by selecting the **"Plan Strain"** option.

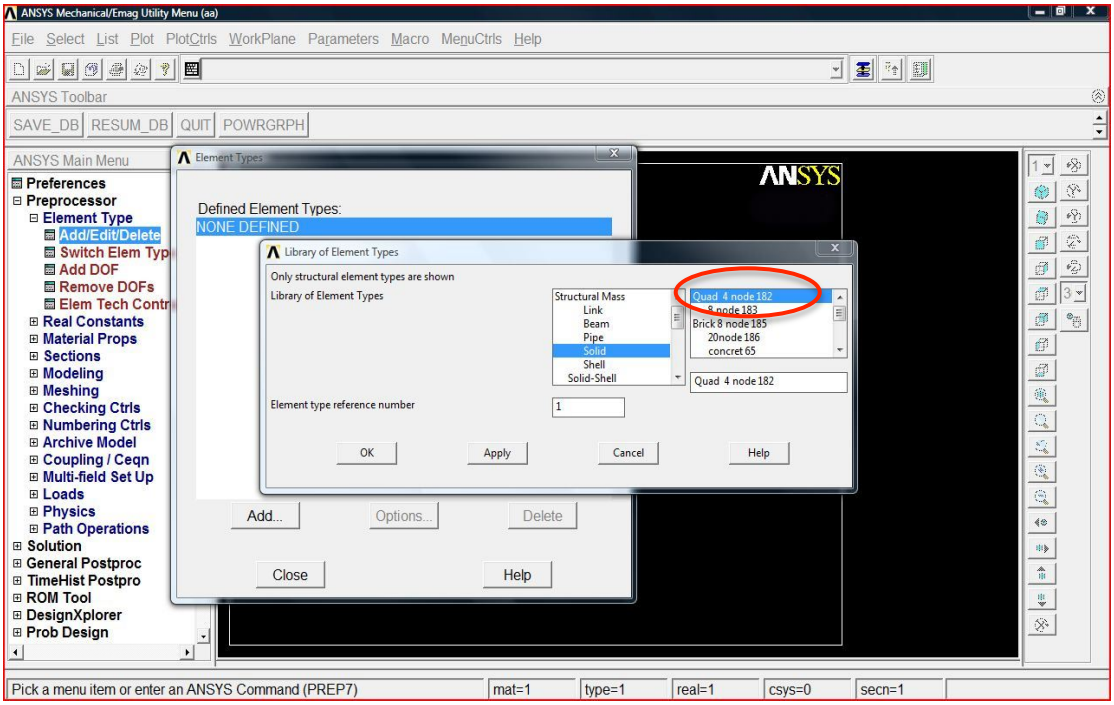


Figure 3.3.2: Elements type's choice

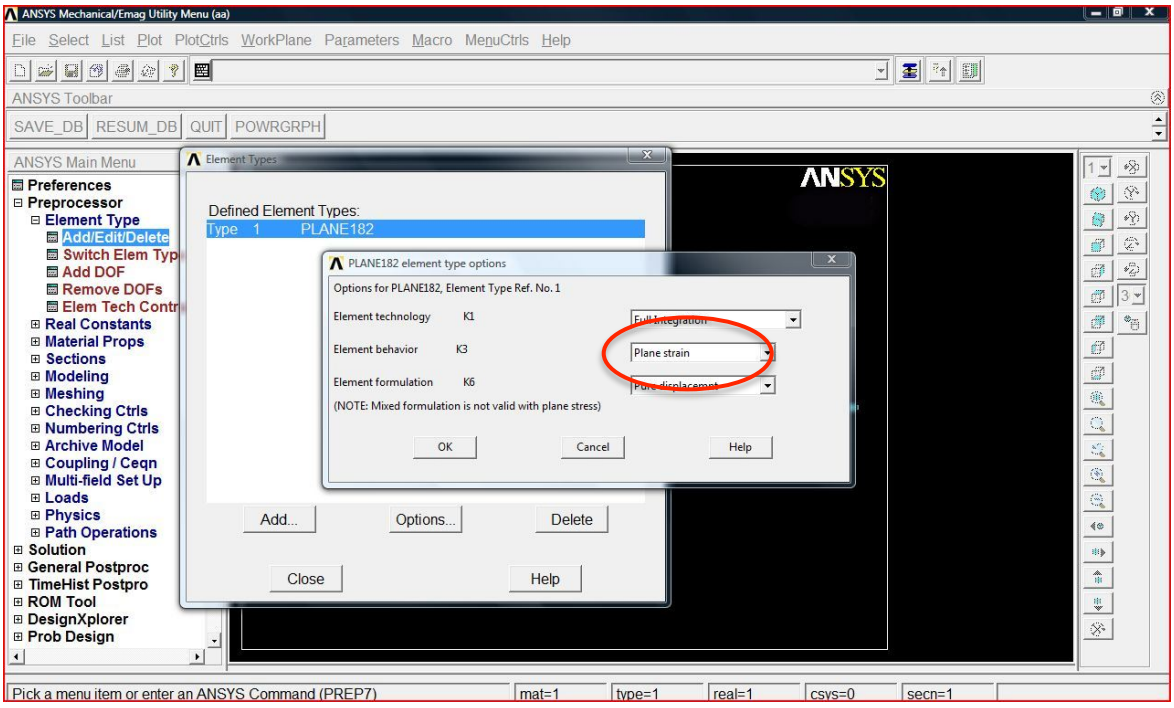


Figure 3.3.3: Elements type's settings – Plane Strain option

It was time to set up the physical properties of the material that the plate would be made of, like the Poisson's ratio, the Young's modulus and the density. In order to define the plate's material, it was done the following:

Main Menu ➔ Preprocessor ➔ Material Props ➔ Material Models

Then, a window appeared so as to select the material type, whose characteristics were **Structural**, **Linear**, **Elastic** and **Isotropic**. Once selected, another window appeared with 2 blanks in order to fill in the Young's modulus (**EX = 121127478933 Pa**) and the Poisson's ratio (**PRXY = 0.301585731837**). Finally, the material's density was specified (**DENS = 4460 kg/m^3**).

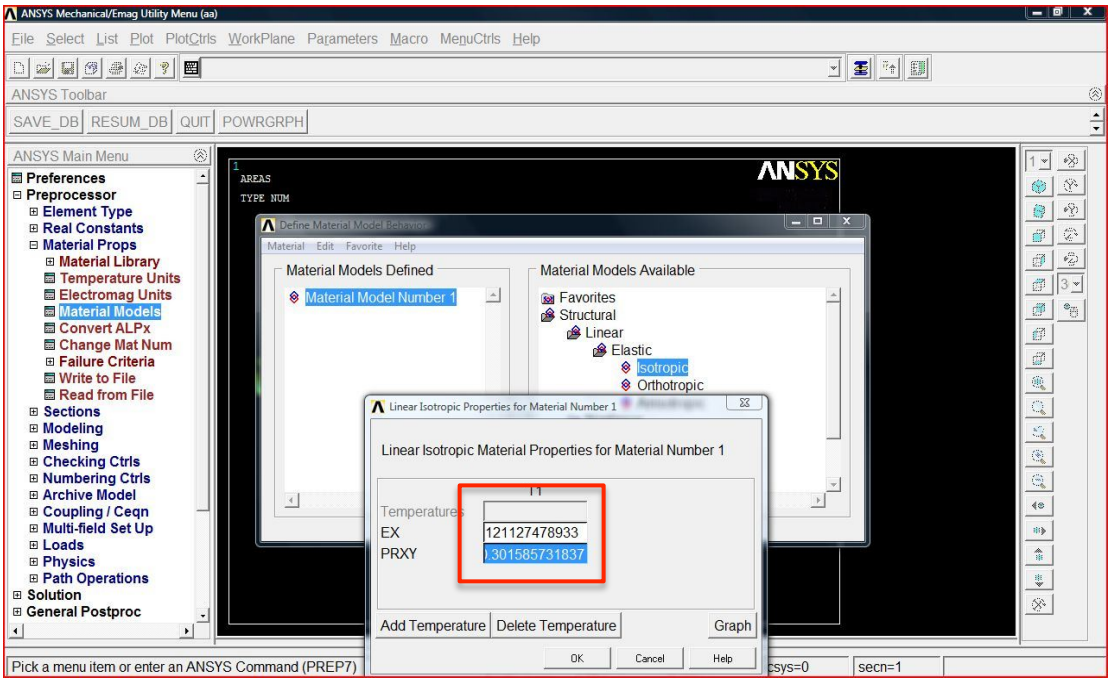


Figure 3.3.4: Physical properties of the titanium

3.4 Meshing Procedure

At that moment, all the parameters that concerned to the geometry and its material were already stated in the software. And so then, the meshing procedure of the plate was ready to get started.

It was very important to know how to mesh the plate, not only the shape of the elements but also their size; the more we decreased the elements size, the plate got

more flexible and the results obtained were more accurate. As it was mentioned in the elements type's section, the geometry would be meshed with square form elements due to the plate's section rectangular form. By meshing the plate's section this way, all the elements would have the same form along the whole section, obtaining consequently more accurate results. This decision pushed to think about how to place the elements' nodes in the subsequent script in order to mesh the plate easily, that would be explained later on. In order to initiate the meshing procedure, it was essential to perform the next operations:

Main Menu ➔ Preprocessor ➔ Meshing ➔ Mesh Tool.

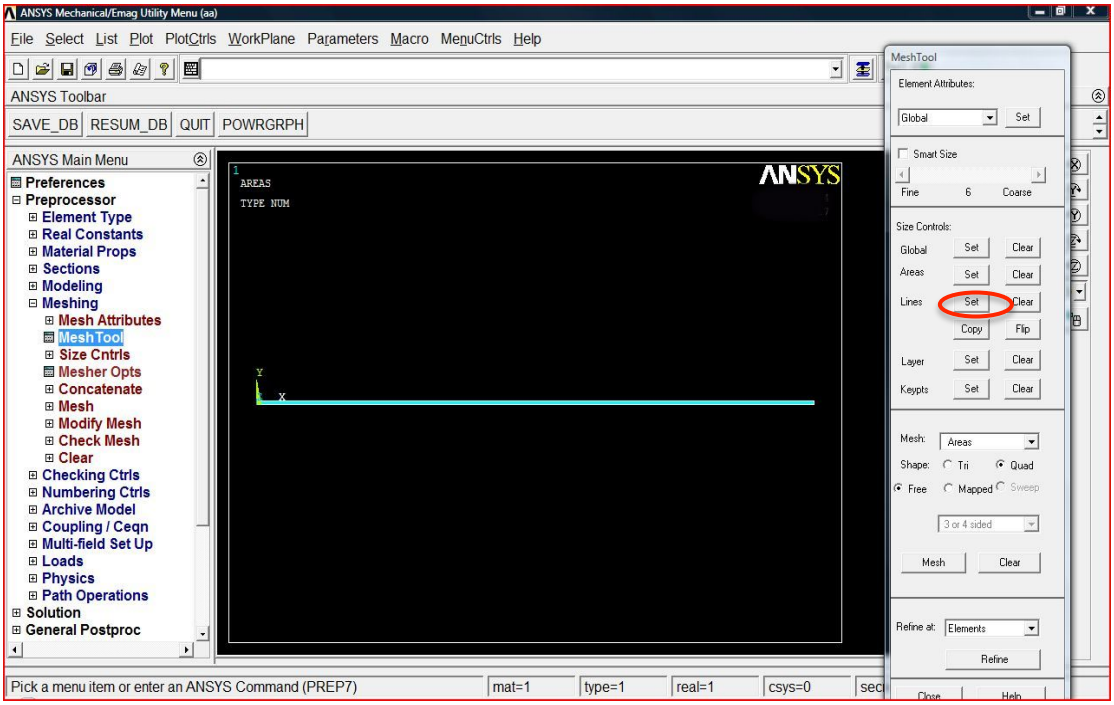


Figure 3.4.1: Beginning of the meshing procedure.

Once finished, another new window raised, this time with the purpose of defining the elements' size. The way to designate it was introducing in ANSYS in how many elements would be each edge divided. The operation that needed to be done was dividing the length of teach edge by the element's size. That is how we did it:

Size Controls ➔ Lines ➔ Set

A little window called **"Element size on Picked Lines"** asked to select the edges that required to be divided. They had to be selected in pairs given that the number of elements in the surface edges was bigger than the amount in the short ones. As it was better to simulate the plate with square elements, it was needed that their size was divisible by the plate's thickness in order to have all the elements a regular form and the same size. Thus, it was though to divide the thickness in 4 elements; less would make the plate be too rigid and dividing it into more pieces would make the simulation take a lot of time to be solved completely. Those were the performed operations:

$$\Delta x_{elem} = \frac{L_{thickness}}{N_{elem.thick.}} = \frac{5\ mm}{4\ elem} = 1.25\ mm/elem$$

$$N_{elem.surf.} = \frac{1000\ mm}{1.25\ mm/elem} = 800\ elem.$$

Once known the size of the elements and the respective number of elements that each edge would be divided in, the procedure continued by selecting them in order to start the division process. At first both surface edges were picked and then the procedure finished with the selection of the lateral ones.

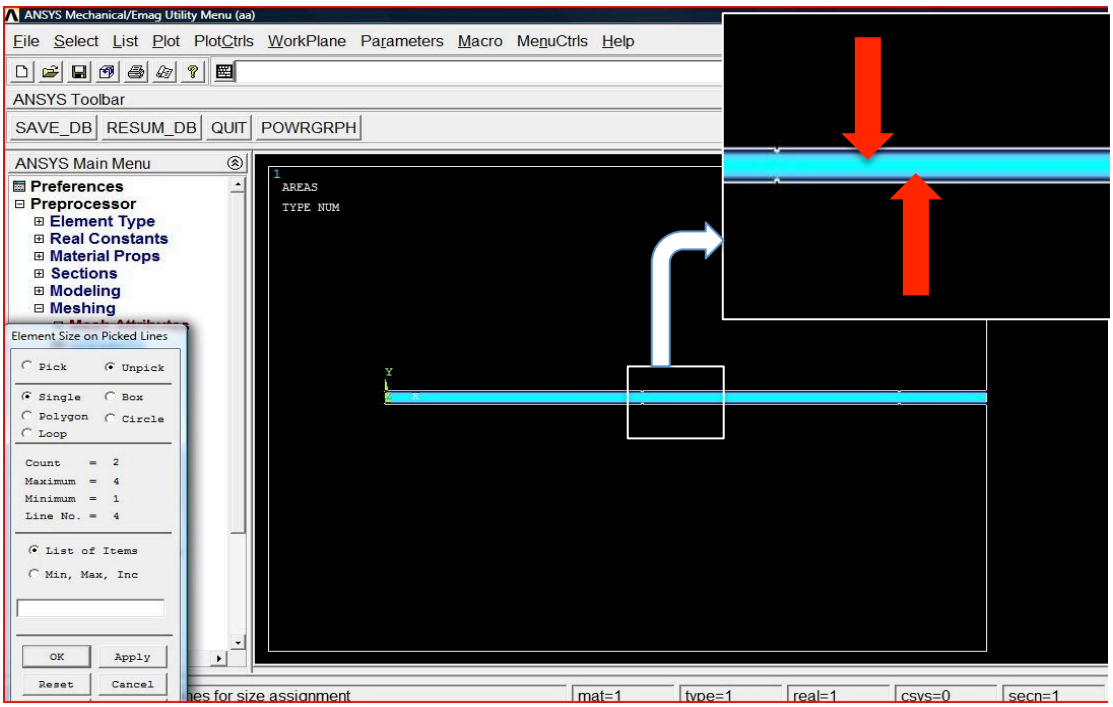


Figure 3.4.2: Surface edges picking process

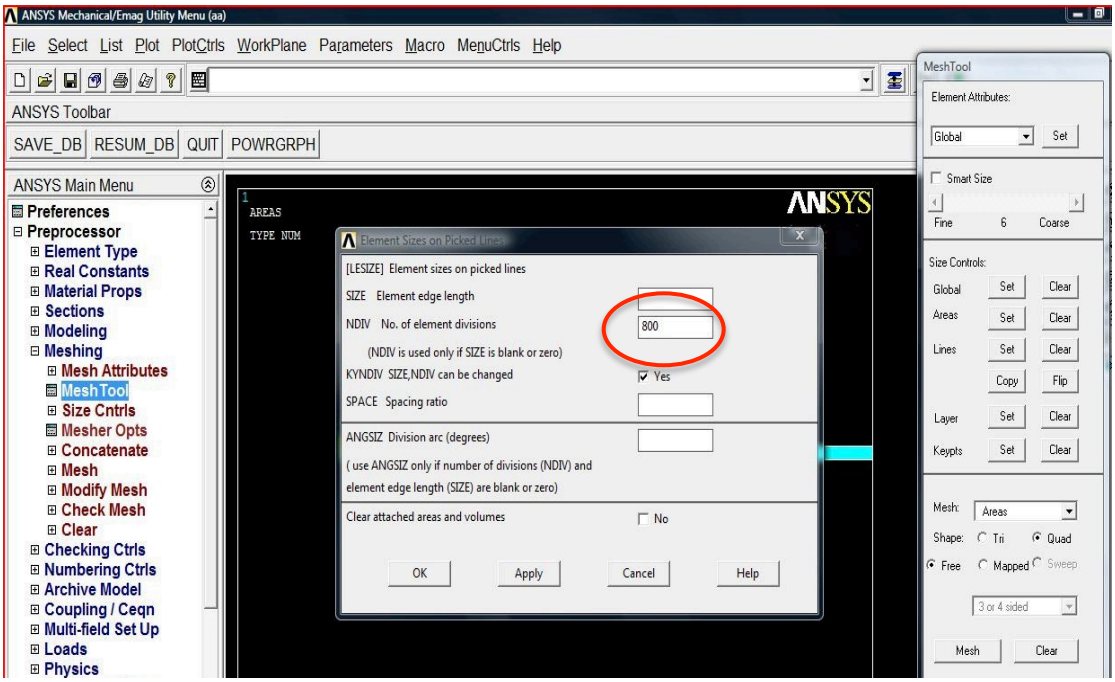


Figure 3.4.3: Surface edges division in 800 elements.

Then the same procedure was repeated so as to pick the lateral edges of the section.

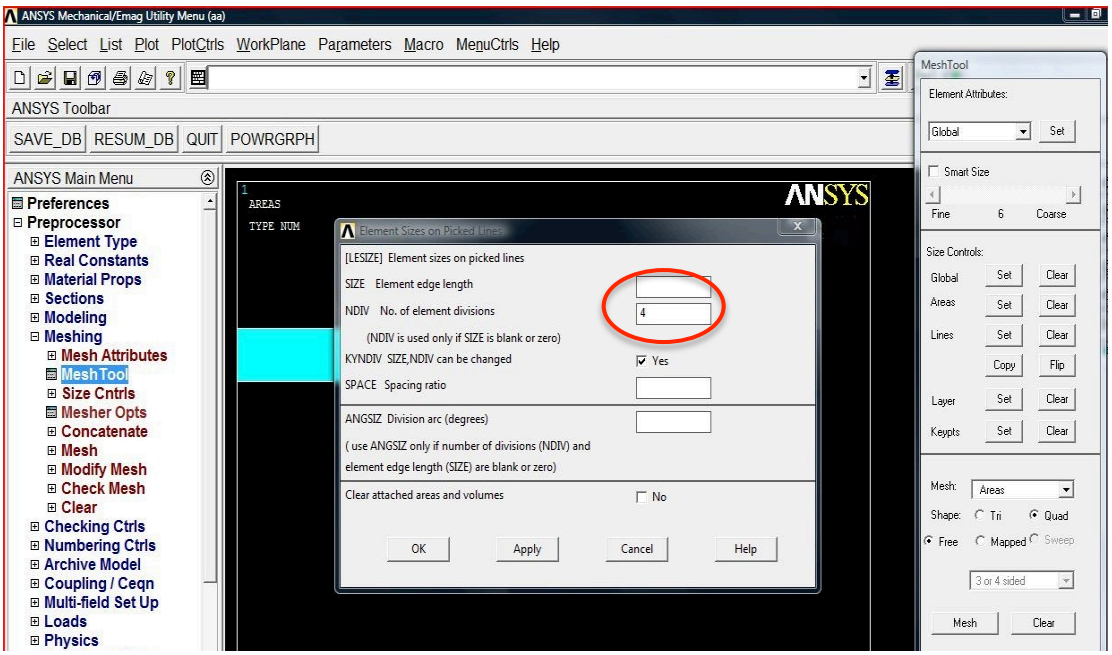


Figure 3.4.4: Lateral edges division in 4 elements

Once the whole perimeter of the plate section was divided in fragments, with the purpose of meshing the plate the area inside the rectangle had to be selected. The window **"Mesh Tool"** was still opened so that the option **"Free"** could be selected in order to mesh afterwards our geometry, which was done by clicking the **"Mesh"** button.

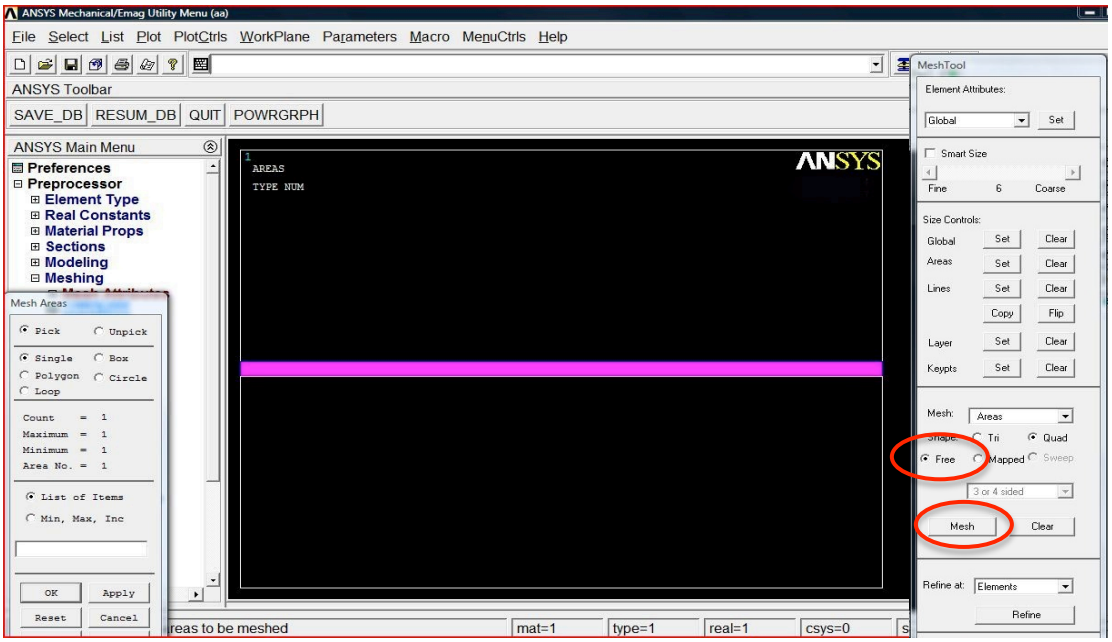


Figure 3.4.5: Area selection

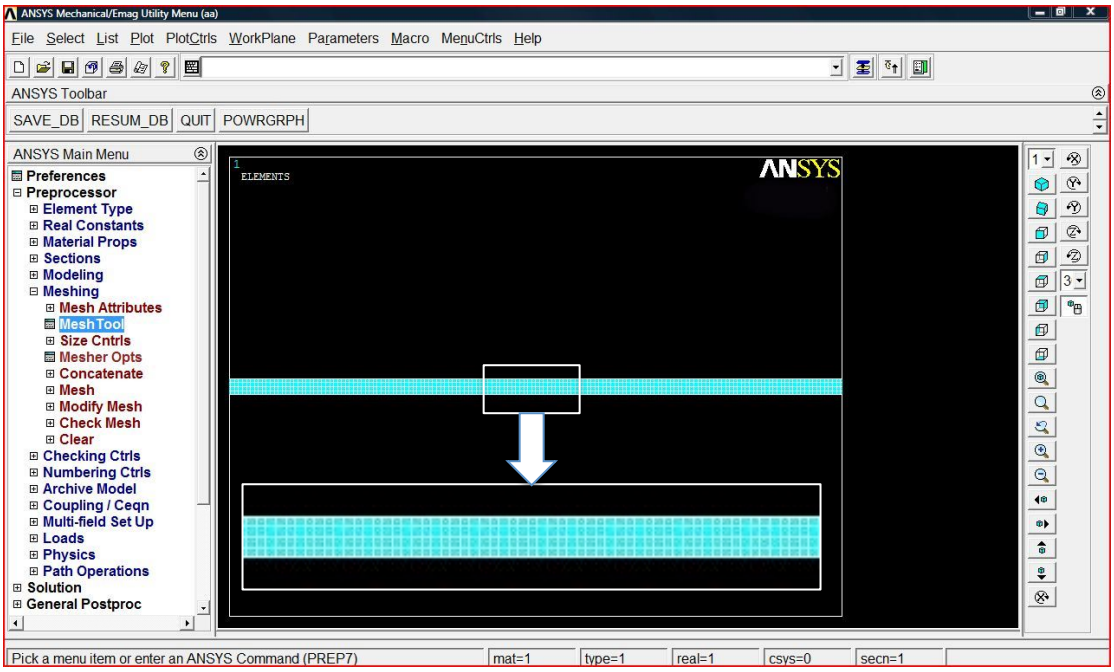


Figure 3.4.6: Figure Meshed

3.5 Function implementation

The geometry was by then completely finished. So, it was time to set up not only the excitation signal but also all its regimes and their corresponding parameters. This was the procedure how to introduce them in ANSYS.

Utility Menu ➔ Parameter ➔ Functions ➔ Define/Edit

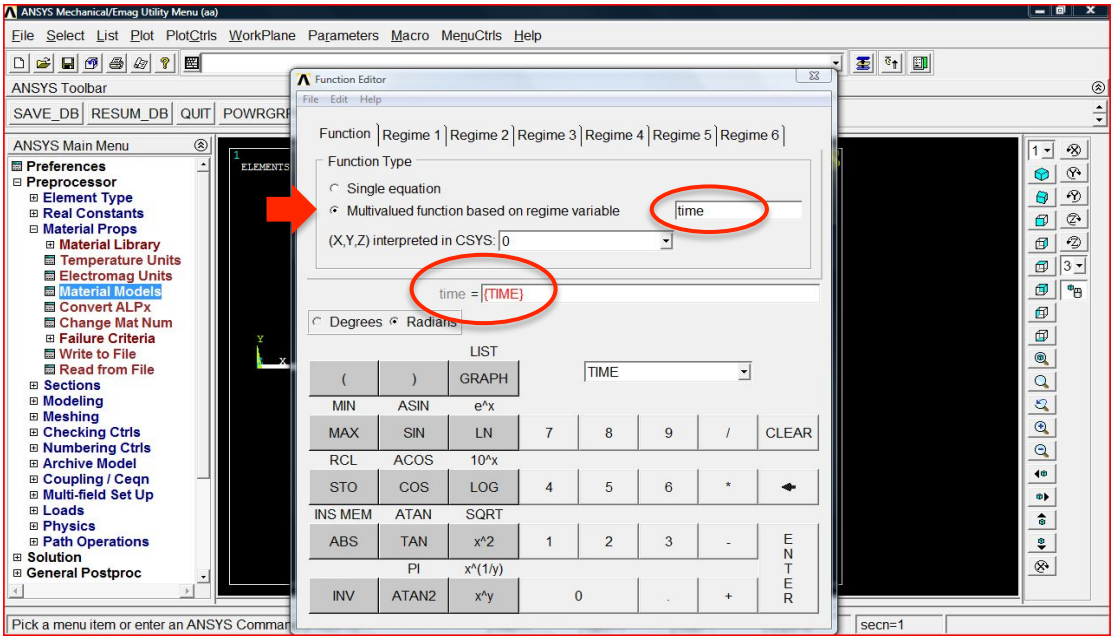


Figure 3.5.1: Function tab. TIME variable definition

In the **"Function Editor"** window it was required to not only define the signal that would be enforced but also set all both regimes excitation and propagation. Accordingly, in the "**Function Type"** blank we needed to choose the option **"Multivalued function based on regime variable".**

The first step to define the function variable (also called Result in the function editor) was choosing a name for it. As it was known that the variable would be the time, it was adopted the option to call it "time" with the purpose of avoiding possible misunderstandings. The name chosen appeared automatically beside the variable blank, which was filled in by choosing the **TIME** variable from a pull-down list.

Next, in the **Regime 1** all the excitation parameters had to be set. It was the place to determine the period of time during which the signal would be implemented and its corresponding equation.

The time during which the plate would be excited was result of an equation that depended of the signal's number of cycles and its frequency. As may be seen, the more we increased the frequency, the excitation period of time got smaller. Here below is the equation whereby the excitation time was obtained.

$$t_{exc} = \frac{n\ cycles}{f_0} = \frac{10\ cycles}{50000\ hz} = 0.0002\ s.$$

As it was said previously, the signal should have a frequency of 50 kHz and 10 cycles, always involved in a Hanning window. Thus, this was the equation we had to introduce in the result blank.

$$\left(0.5 * \left(1 - cos\left(\frac{2 * 3.14 * t}{0.0002}\right)\right)\right) * sin(2 * 3.14 * 50000 * t)$$

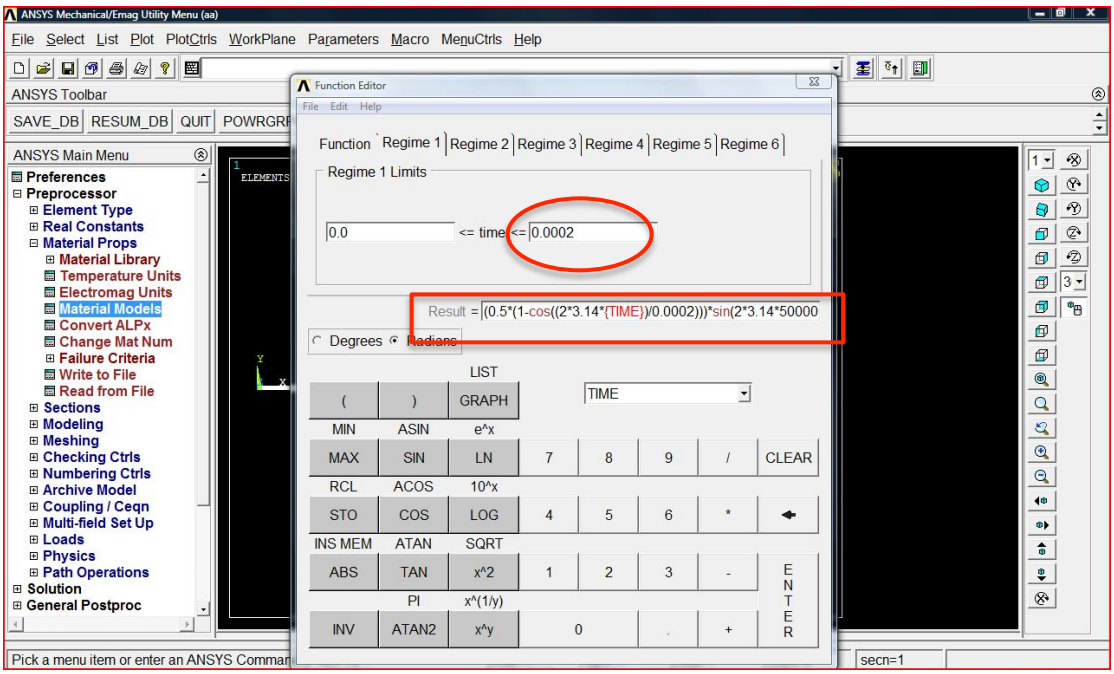


Figure 3.5.2: Regime 1 tab. Excitation. Signal Function Edit.

The **Regime 2** tab was where the propagation preferences had to be defined. During this period of time the signal could not be applied, so the TIME variable had to be deleted. As the function blank could not be left empty, the way to remove it was multiplying the TIME by 0. Furthermore, the ending simulation time required to be determined. That period of time needed to be long enough to allow the signal be propagated until 1 meter far from the excitation point, where the node that was of our

interest was placed. To choose that ending time a reference software called Disperse was employed in order to see how long took to the signal to arrive at the mentioned point. With it, both displacements' graphs excitation and propagation were obtained noticing that the propagation ended approx. 0.00072 s. after the signal excitation. However an ending time of 0.001 s was chosen instead; this period was long enough not only to allow the signal reach the point of interest but also to comprehend how it behaved after arriving to that specific node.

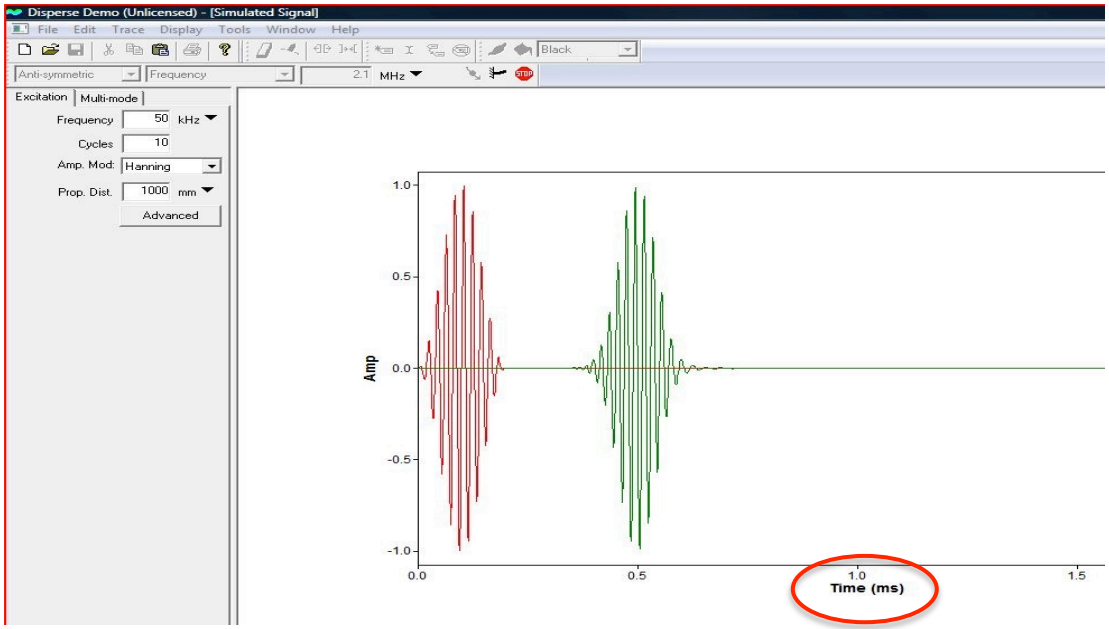


Figure 3.5.3: Disperse results. Ending propagation period time.

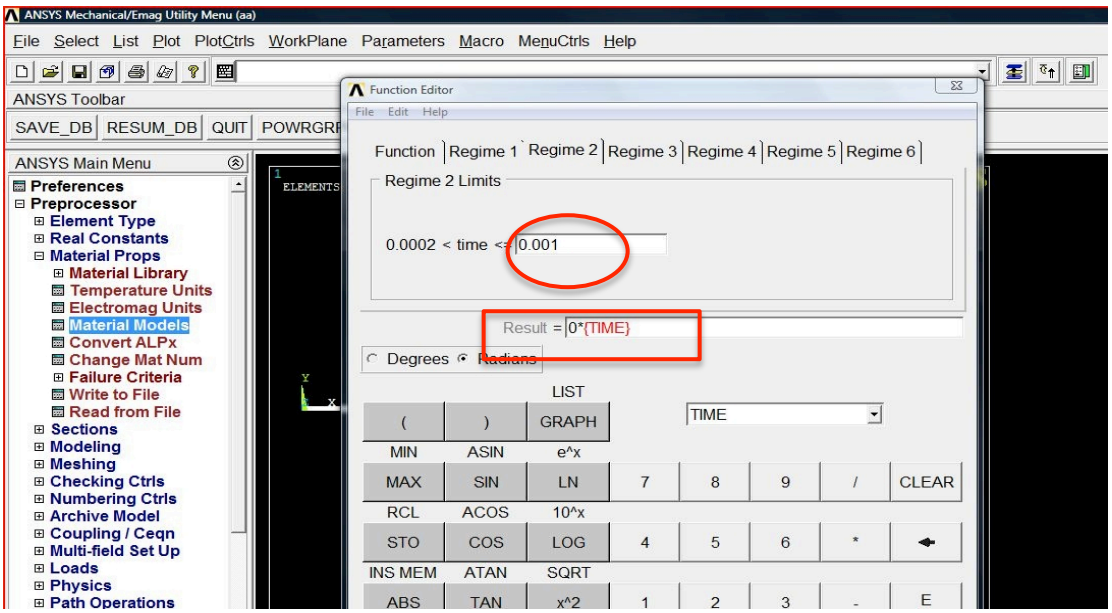


Figure 3.5.4: Regime 2 tab. Propagation.

3.6 Transient Analysis

Afterwards, when all the preferences were introduced in ANSYS, the function was saved in order to use it at the time of implementing the signal at the corresponding node.

In order to employ the previously stored function, it was required to create a new analysis. It needed to be transient to set correctly both the boundary and the initial conditions. This procedure needed to be followed.

Main Menu ➔ Solution ➔ New Analysis ➔ Transient

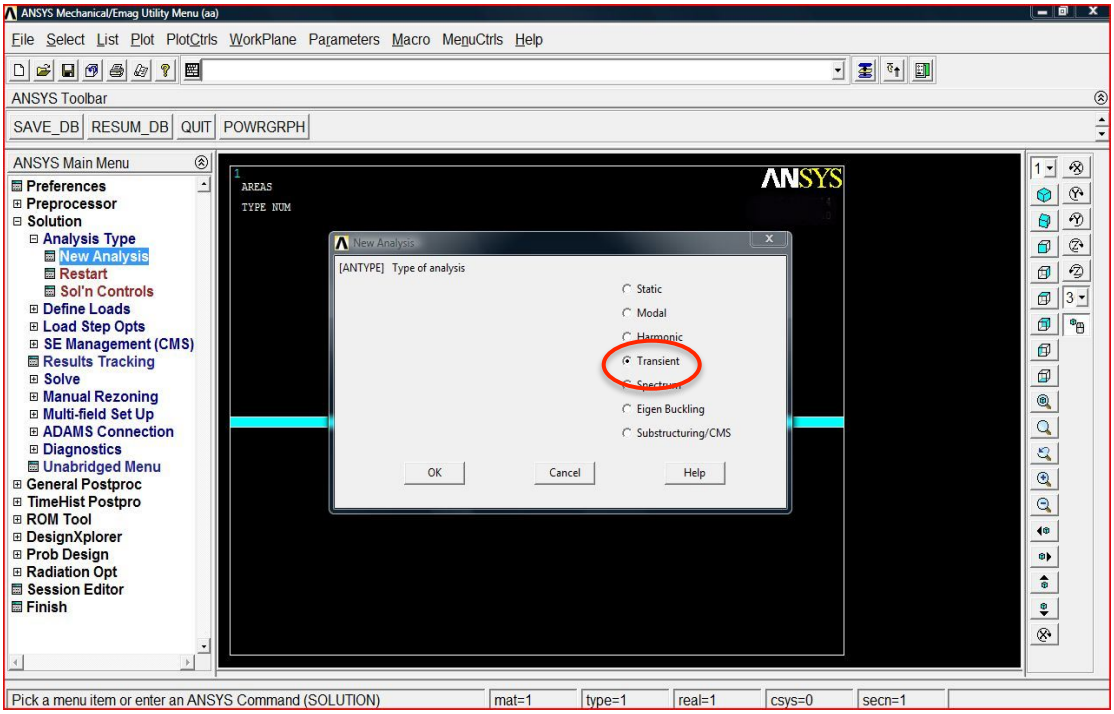


Figure 3.6.1: Transient analysis setting

Once chosen this type of analysis, a new emergent window appeared so as to select the solution type. As a very detailed simulation resolution for the whole plate was desired, the "**Full"** option was elected accordingly. That way, it was ensured that a displacement for every node of the plate would be obtained.

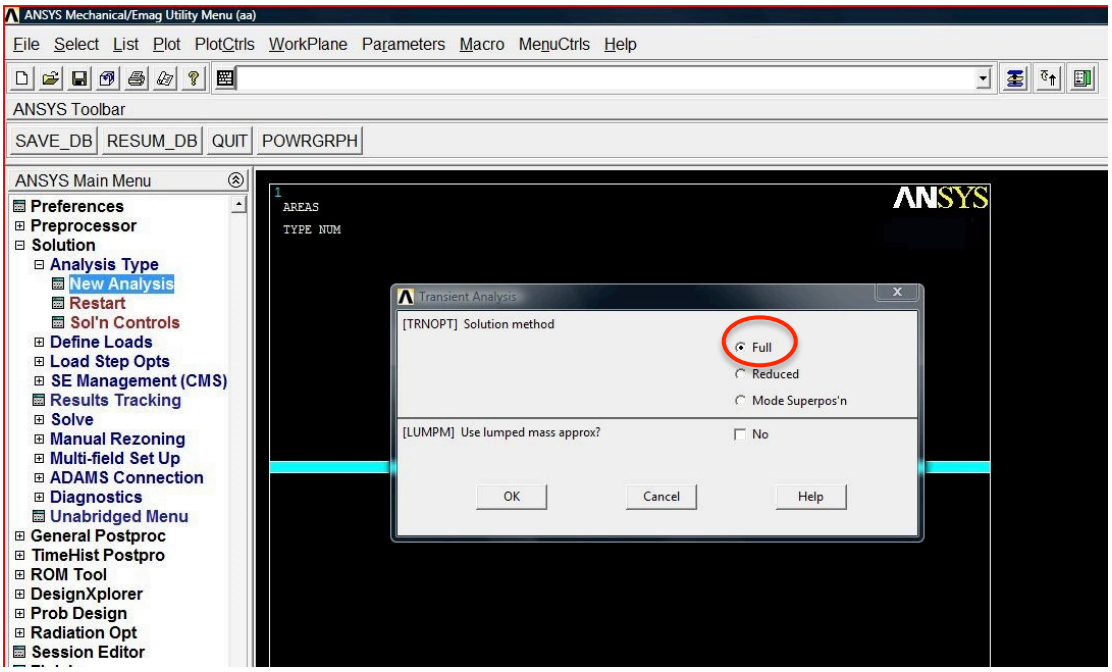


Figure 3.6.2: Full option selection in the solution method menu.

The following step was to fix al least one node of the plate so as to have non-singular stiffness matrix. This matrix represents the system of equations that must be solved in order to ascertain an approximate solution to the differential equations proposed by the finite elements method. In order to have the least influence in the simulation results, we decided to fix one of the plate's corners nodes, more specifically one of the two nodes located in the same side edge where the signal was implemented. The easiest and quickest way to perform the task was to pick directly the suitable node from the graphics screen.
Those are the operations carried out to fix the node.

Main Menu ➔ Solution ➔ Define Loads ➔ Apply ➔ Displacement ➔ On nodes

Once picked, a new window emerged with the purpose of selecting which of the degrees of freedom we needed to block. As the node needed to be completely fixed, the option "**All DOF (All Degrees Of Freedom)"** was chosen.

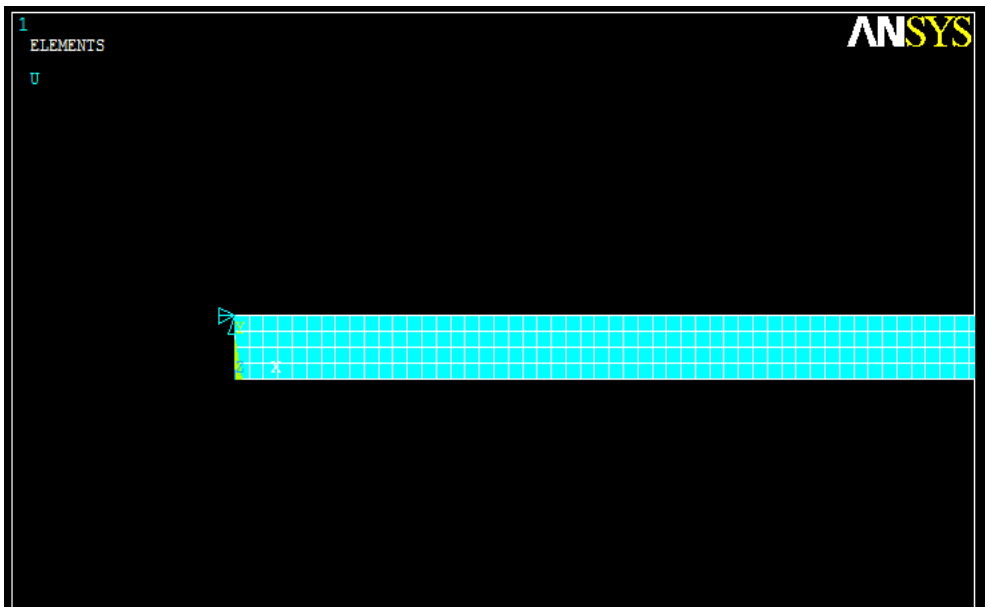


Figure 3.6.3: Node's fixation (All degrees of freedom)

3.7 Signal enforcement

Once the node was fixed, the signal previously saved was read and implemented at the corresponding node, which needed to be located in the middle of the plate's thickness in order to avoid having the modes both A0 and S0 in the y-direction displacements; in case of being in any of the two plate's surfaces, both modes would have appeared in the displacements graph. Moreover, it was required to place the node in one of the vertical edges of the plate. So that was how it was done:

Utility Menu ➔ Parameters ➔ Functions ➔ Read from file

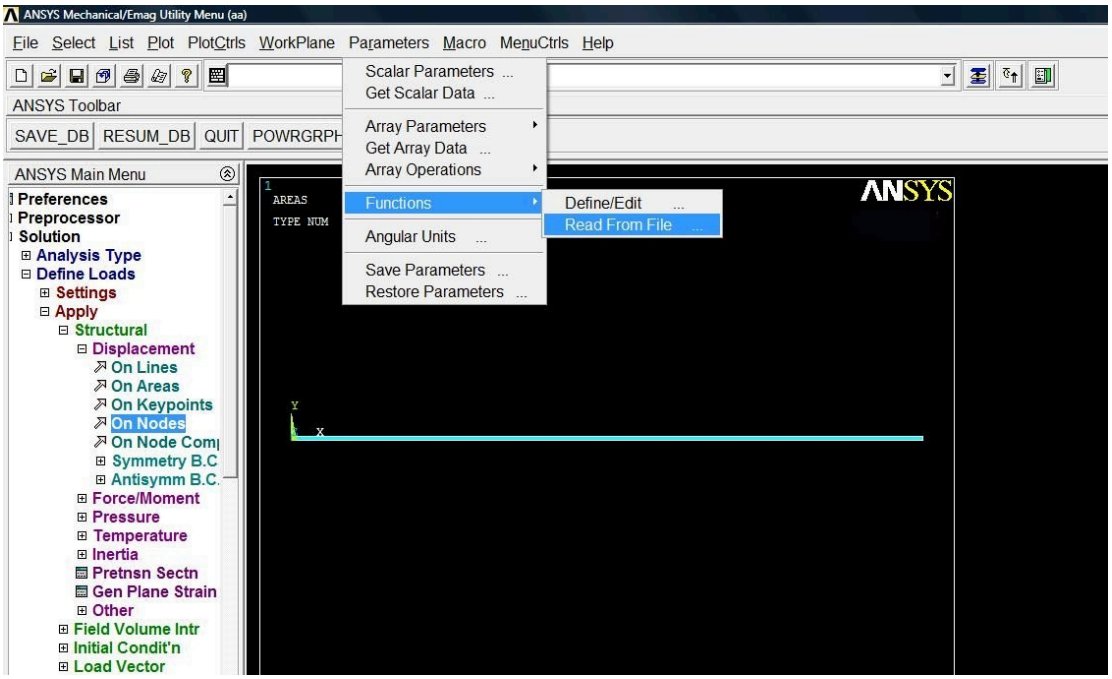


Figure 3.7.1: Function reading from the file created.

A new window called **"Function loader"** appeared where the file saved previously under the name filename.func was supposed to be selected. Once the function was chosen, a name for the table that would be used in the force determination had to be written. As before and in order to avoid confusions, the called it "hanning", corresponding to the window that enveloped the signal.

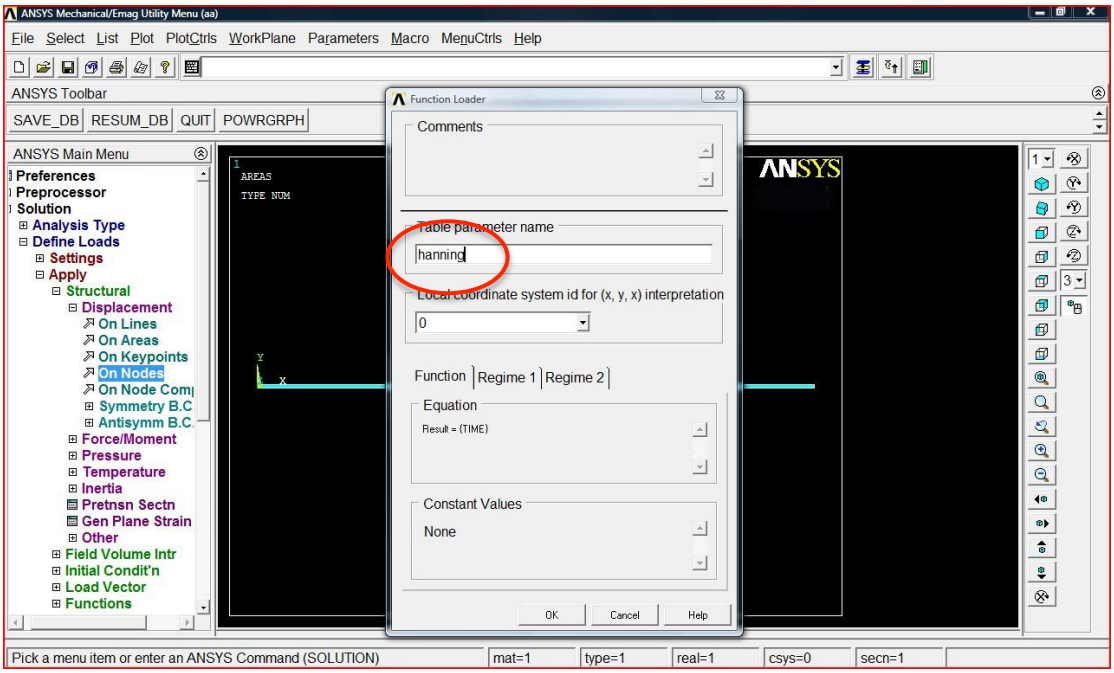


Figure 3.7.2: Table parameter name

Then it was time to enforce the force on the plate; all the values from the table early written needed to be implemented following the y-axis direction in the right node. Here below is shown how it was done.

Main Menu ➔ Solution ➔ Define Loads ➔ Apply ➔ Structural ➔ Displacement ➔ On nodes

Then, a new emergent window called “Apply F/M on Nodes” appeared in order to pick the node where the function was supposed to be implemented, following the instructions beforehand cited. The specified node was located under the coordinates X=0 mm and Y=2.5 mm, whose id was 1607.

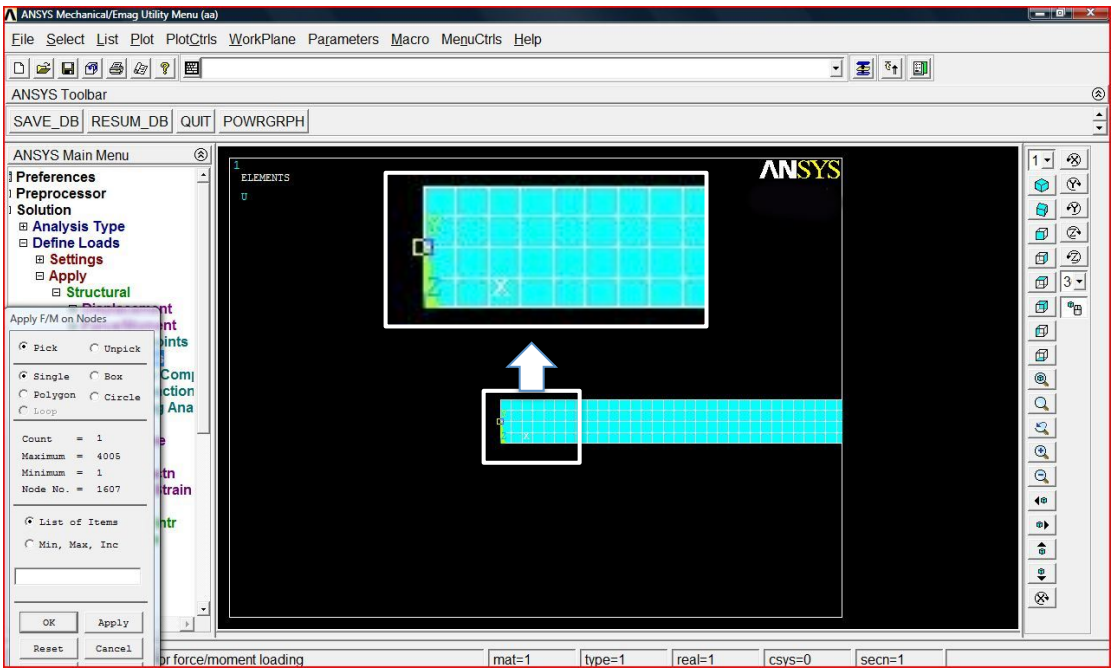


Figure 3.7.3: Node Selection. Signal Excitation.

Therefore, another window appeared where the force direction and its value had to be set. From the very beginning we determined that our object of study were mainly the nodes' displacement in the y-axis direction, thus the force had to be enforced in the same direction. In the same window, there was a blank where one of the options "apply a constant force" or "apply a created function" (always in a table form as done previously) could be chosen. As it was needed to use the table created some steps ago, the option **"existing table"** was selected. Once confirmed these preferences, another window appeared where the "hanning" table (where all the function values that would be implemented on the plate were written) was supposed to be selected.

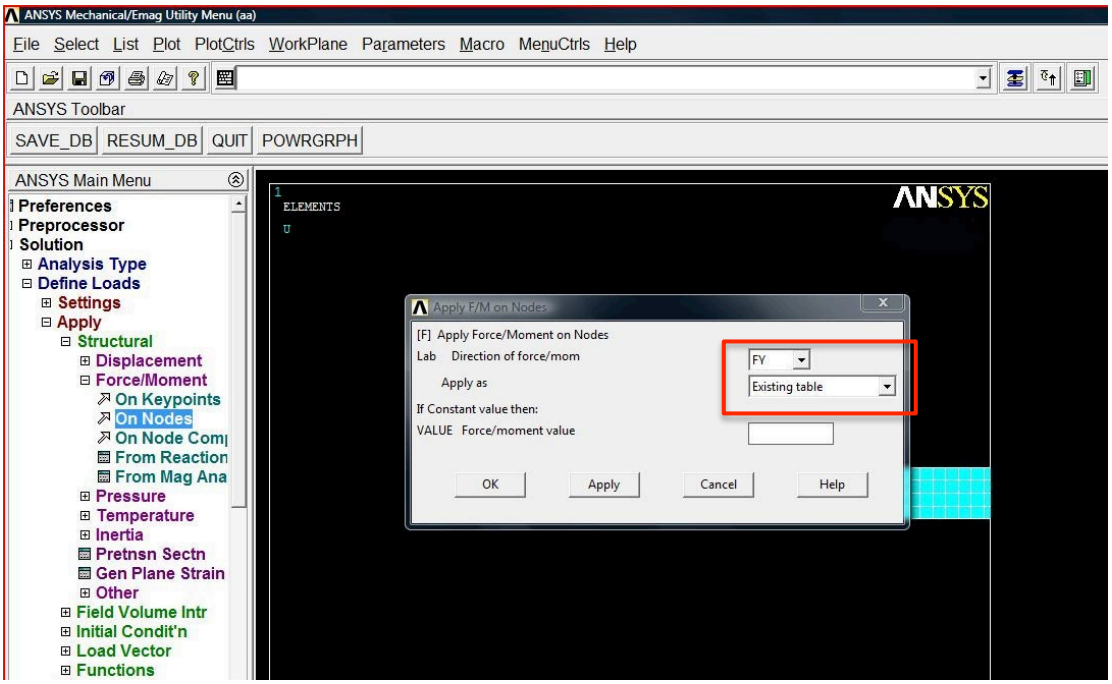


Figure 3.7.4: Force application settings

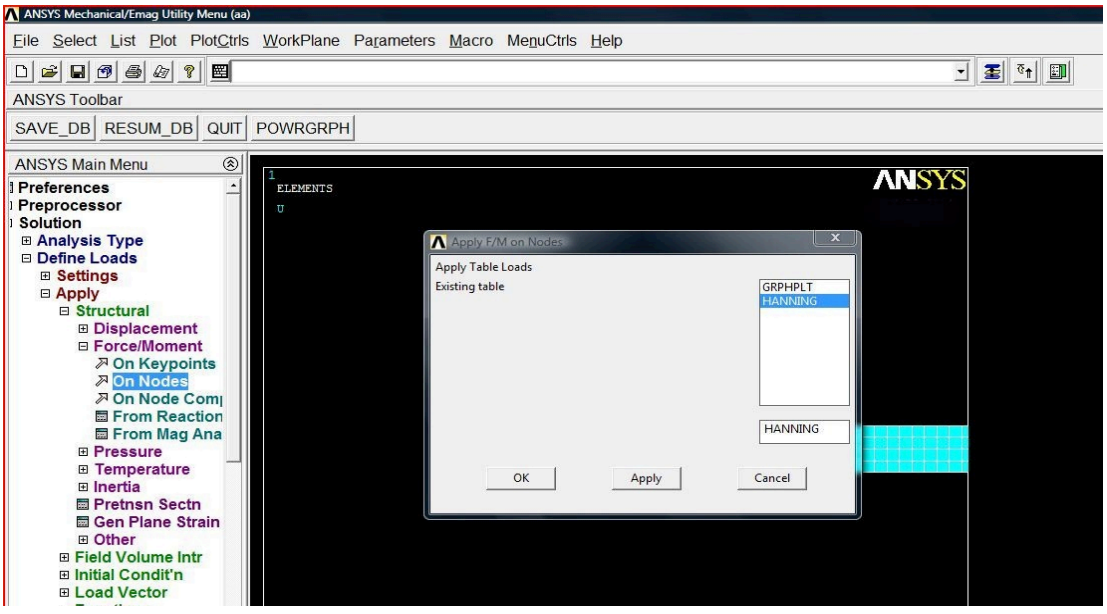


Figure 3.7.5: Value table Selection

3.8 Initial Conditions' settings.

At that moment, the boundary conditions were defined, so what remained was to impose the initial conditions of the plate to all the nodes that had been created. To make it possible, we followed the procedure described below.

Main Menu ➔ Solution ➔ Define Loads ➔ Apply ➔ Initial Condit'n ➔ Define➔ Pick All

An emergent window appeared so as to pick the area in which we were working on and define its initial conditions. So once picked the plate section area, another window raised with the purpose to choose which of the DOF constraints were defined. As all the nodes were free except one, which was already fixed, the option **"All DOF"** was selected from the list.

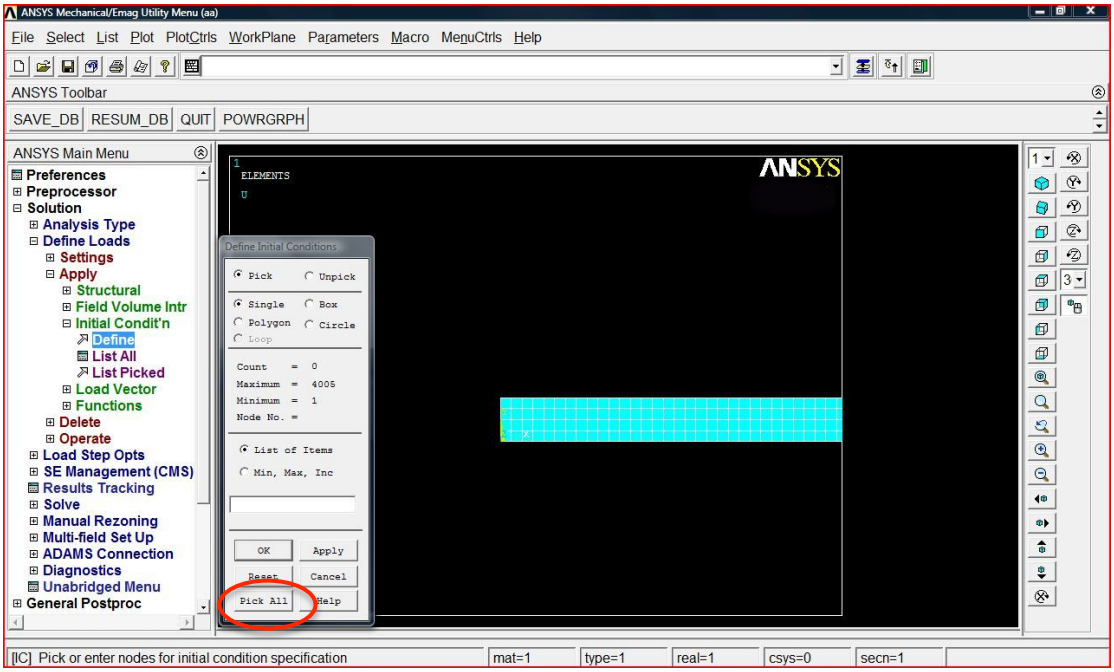


Figure 3.8.1: Definition of the initial conditions.

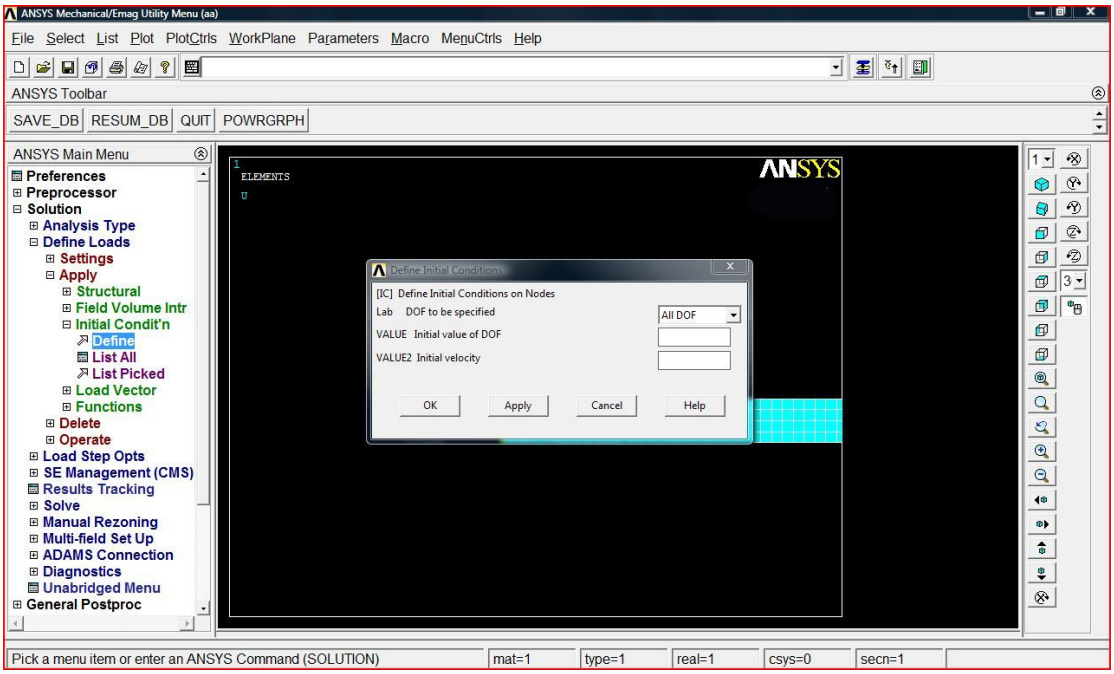


Figure 3.8.2: Selection of the option “All DOF”.

3.9 Solution timing and output frequency.

The simulation was ready to define the output’s frequency and the solution time step; in other words, the period of time that elapses between a solution taken by ANSYS (UX/UY) and the immediately subsequent. Here below is the formula that gave as result the biggest time step that fitted for this simulation. As may be seen, it depends of the elements’ size and the maximum phase velocity, which was obtained with the Disperse software.

$$\Delta t = \frac{0.8 * \Delta x_{elem.}}{V_{max}} = \frac{0.8 * 2.5\,mm}{1474\frac{mm}{ms} * 10^3} = 1.3 * 10^{-6} \cong 1 * 10^{-6}\,s.$$

In this formula, the term $\Delta x_{elem.}$ corresponds to the size of the elements while the V_{max} term means the maximum phase velocity of a 50kHz signal in the A0 mode.
The procedure to introduce all preferences in ANSYS was the next one:

Main Menu ➔ Solution ➔ Load Step Opts ➔ Unabridged Menu

After this action, the menu changed to its full version, allowing then to set all the time and frequency settings. With the whole drop-down menu visible, the procedure required to do the succeeding operations.

Time/Frequenc ➔ Time-time step

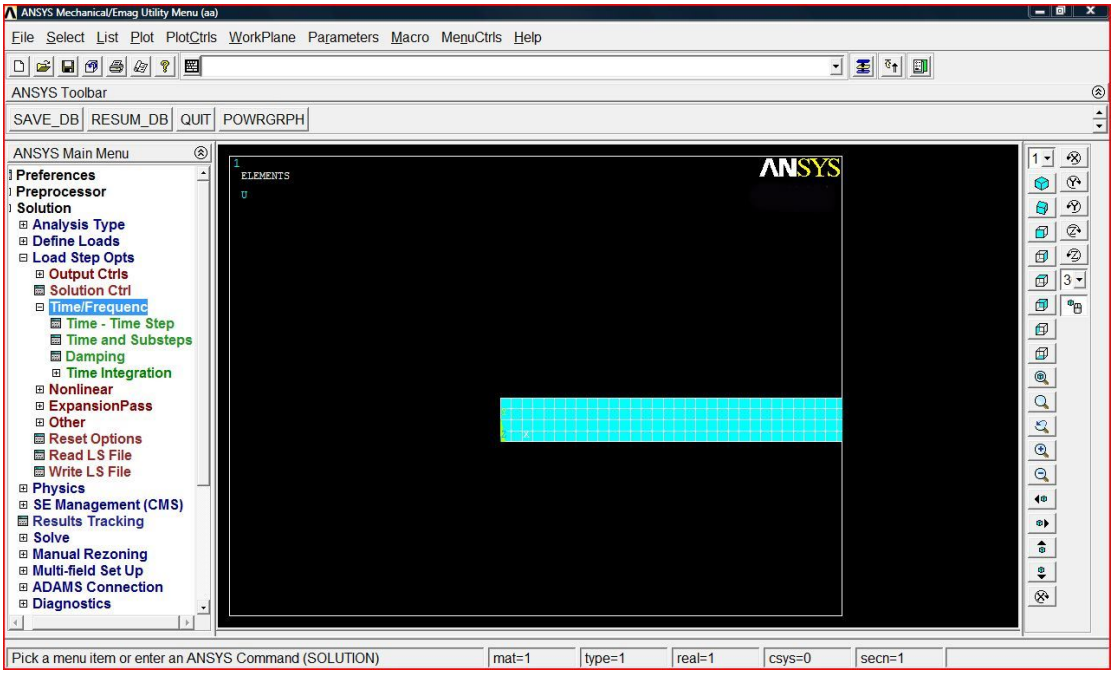


Figure 3.9.1: Time – Time Step menu

An new display called **"Time and Time Step Opts"** appeared where the time step options were supposed to fill in their corresponding blanks, always following the instructions described below:

Time at the end of Load Step = t_{final} = 0.001 s

Time step size = 0.5*Δt = 0.0000005 s

KBC = Stepped

Automatic time stepping = OFF

Minimum time step = 0.1*Δt = 0.0000001 s

Maximum time step = Δt = 0.000001 s

Use previous step size = YES

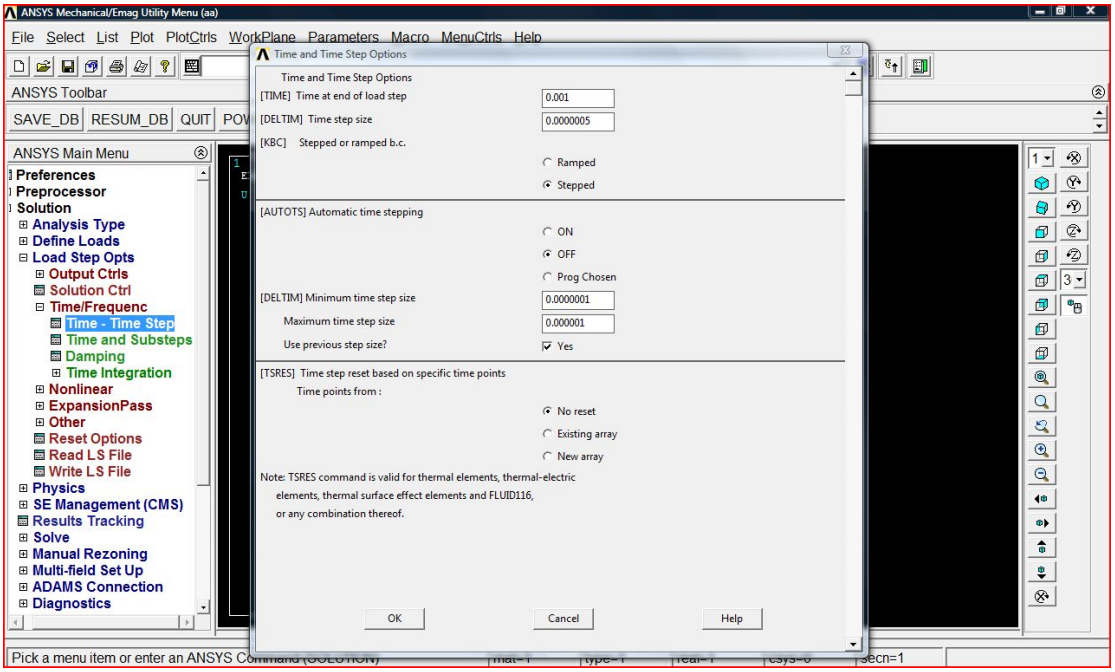


Figure 3.9.2: Solution timing and output frequency settings.

The reason why the Loads had to be stepped was because the whole signal needed to be enforced from the first substep. Otherwise, the whole signal would have been just applied at the last substep, which would have offered us a wrong simulation. The explanation is shown in the next figure.

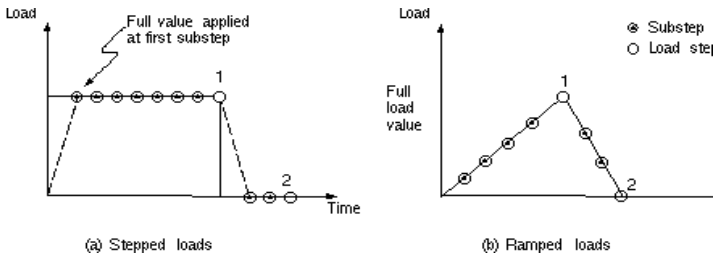


Figure 3.9.3: Difference between Stepped Loads and Ramped Loads.

As it was desirable to have maximum control over the simulation, the automatic time stepping was disabled. This way, it was ensured that the software would obtain a valid solution for every period of time introduced by ourselves.

The configuration of our simulation was almost finished but before it was needed to determine which of the items would be written in the database and the results file. It could have chosen just to obtain the nodal degrees of freedom such as UX or UY,

but it was chosen instead to obtain all items just in case the study of another magnitude was needed. The way it was done was by marking the option **"All items"** in the **"Items to be controlled"** blank.

If this were not enough, the output frequency had to be set too. It was required to write a result for every completed substep as the simulation time progressed, the option **"FREQ file write frequency: Every Nth substep"** had to be chosen. So as to obtain the maximum amount of points for our simulation results, a small number of substeps (N) needed to be stated. For this simulation, 2 substeps were selected in order to have 1000 points for the displacement graphs. Here below is presented the equation that gives as result the number of points that would take part of the displacement's graph.

$$\boldsymbol{points = \frac{t_{sim}}{N * t_{step\ size}} = \frac{0.001\ s}{2 * 5 * 10^{-7}\ s} = 1000\ points}$$

In this operation, $\boldsymbol{t_{sim}}$ corresponded to the total simulation time, N meant the number of substeps chosen and $\boldsymbol{t_{step\ size}}$ was the time previously described. It was impossible to have 2000 points because in case of choosing just one substep, ANSYS arrived to an error that could not be solved at all. Here is shown how it was achieved:

Main Menu ➔ Solution ➔ Load Step Opts ➔ DB/Results File

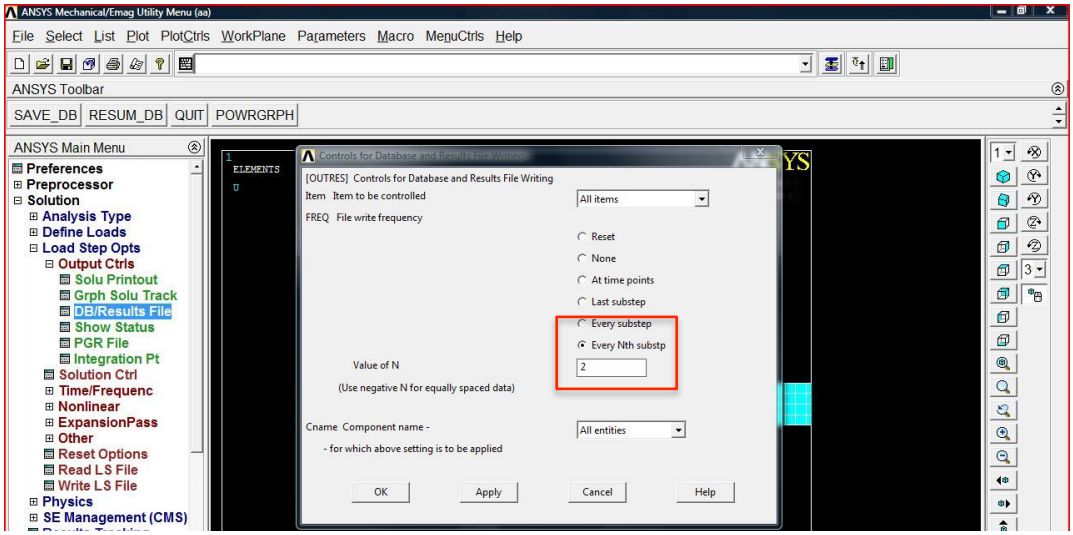


Figure 3.9.4: File write frequency (1 sol / 2 substeps)

3.10 Simulation analysis

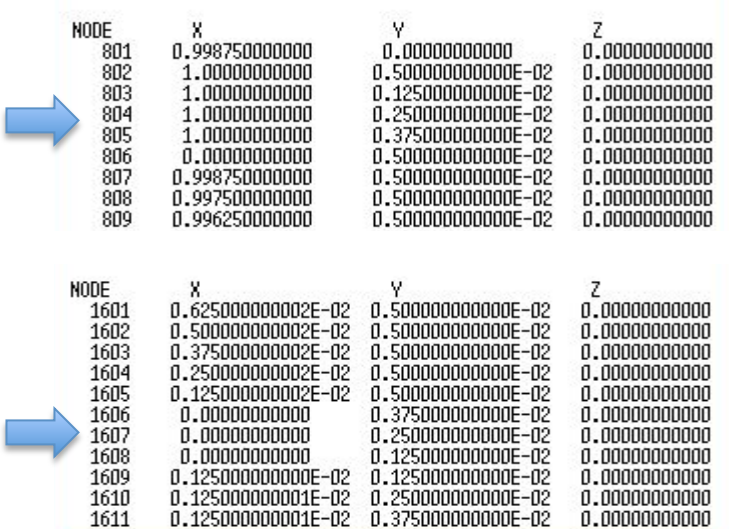
It was high time to check if the procedure done was correct or not. In order to verify if the simulation could be coded, it was precise to start the simulation by completing these operations.

Main Menu ➔ Solution ➔ Solve ➔ Current LS.

The simulation kept on working until the message **"Solution is done!"** appeared in the display.

The way to validate if the solution found was correct was obtaining the displacements of both the excitations and the analysis nodes and comparing them with the solutions obtained with Disperse by using the Matlab software. Before checking the corresponding solutions, it was necessary to know the nodes' id they required to be analysed with the purpose of checking easily their displacements. As their coordinates were known (X=0.0 mm Y=2.5 mm; X=1000 mm Y=2.5 mm) so with a node list it was easy to identify them. That list was obtained by doing these stages:

Utility Menu ➔ List ➔ Nodes ➔ Coordinates Only



NODE	X	Y	Z
801	0.998750000000	0.00000000000	0.00000000000
802	1.00000000000	0.500000000000E-02	0.00000000000
803	1.00000000000	0.125000000000E-02	0.00000000000
804	1.00000000000	0.250000000000E-02	0.00000000000
805	1.00000000000	0.375000000000E-02	0.00000000000
806	0.00000000000	0.500000000000E-02	0.00000000000
807	0.998750000000	0.500000000000E-02	0.00000000000
808	0.997500000000	0.500000000000E-02	0.00000000000
809	0.996250000000	0.500000000000E-02	0.00000000000

NODE	X	Y	Z
1601	0.625000000002E-02	0.500000000000E-02	0.00000000000
1602	0.500000000002E-02	0.500000000000E-02	0.00000000000
1603	0.375000000002E-02	0.500000000000E-02	0.00000000000
1604	0.250000000002E-02	0.500000000000E-02	0.00000000000
1605	0.125000000002E-02	0.500000000000E-02	0.00000000000
1606	0.00000000000	0.375000000000E-02	0.00000000000
1607	0.00000000000	0.250000000000E-02	0.00000000000
1608	0.00000000000	0.125000000000E-02	0.00000000000
1609	0.125000000000E-02	0.125000000000E-02	0.00000000000
1610	0.125000000001E-02	0.250000000000E-02	0.00000000000
1611	0.125000000001E-02	0.375000000000E-02	0.00000000000

Figure 3.10.1: Node's list. Coordinates of the studied nodes.

When the nodes numbers were known, the attained displacements could be checked if they were correct or not. The easiest procedure to obtain those displacements was by writing directly the node's id in the Variable viewer by following the next steps.

Main Menu ➔ General Postcroc ➔ TimeHist Prospro ➔ Variable Viewer

Afterwards a window called **"Time History Variables"** appeared, in which we could select every node of the plate and study all its degrees of freedom offered by ANSYS software, in this case the y component of displacement. With the node list obtained in the previous step, the node number could be written in the corresponding blank instead of picking it directly from the plate's figure, which was a more tedious process. Once shown those displacement results both results files were saved in order to be compared afterwards with the reference ones. The way to obtain those results was by continuing the procedure previously started:

Add Time History Variable ➔ Nodal Solution ➔ DOF Solutions ➔ Y-Component of displacement. ➔ Node id ➔ Save

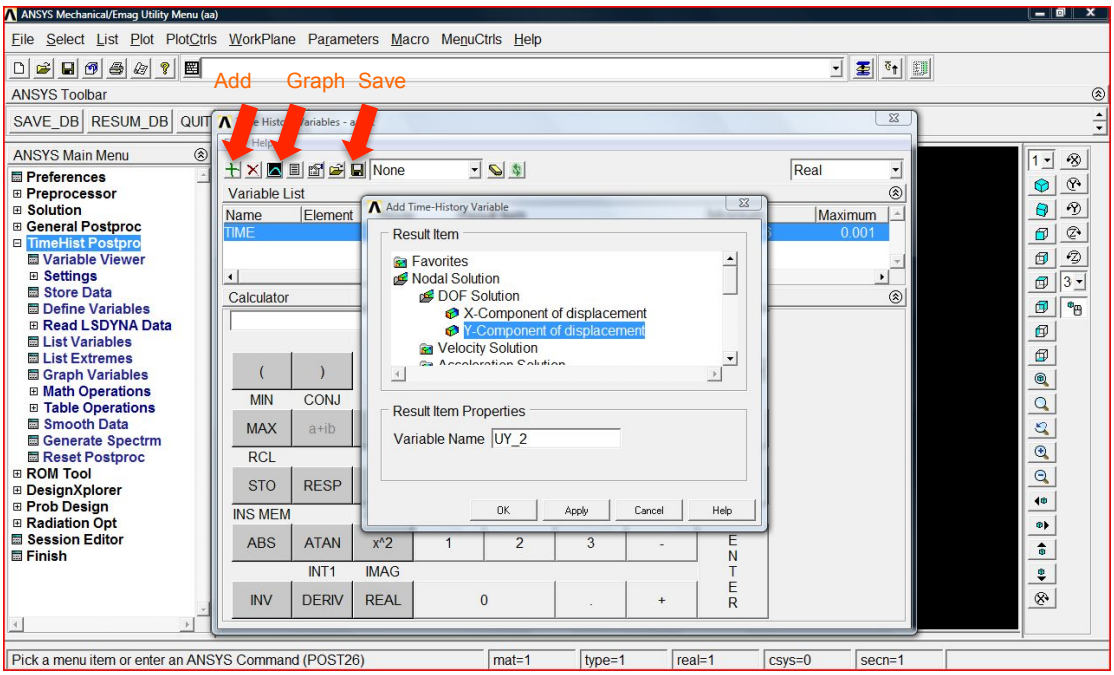


Figure 3.10.2: Time-History Variable selection (Y-axis nodes displacement)

This was the result obtained the node where the signal was implemented. On first thought, the results reached did not coincide with the ones we took as reference. The simulation seemed to be correct but unexpectedly we obtained a signal rebound. That fact led to think about the possibility that the mistake was made because of the short size of the plate. As we created an object of 1 m long the signal had had time

enough to reach the end of the plate and consequently return to its origins creating a rebound.

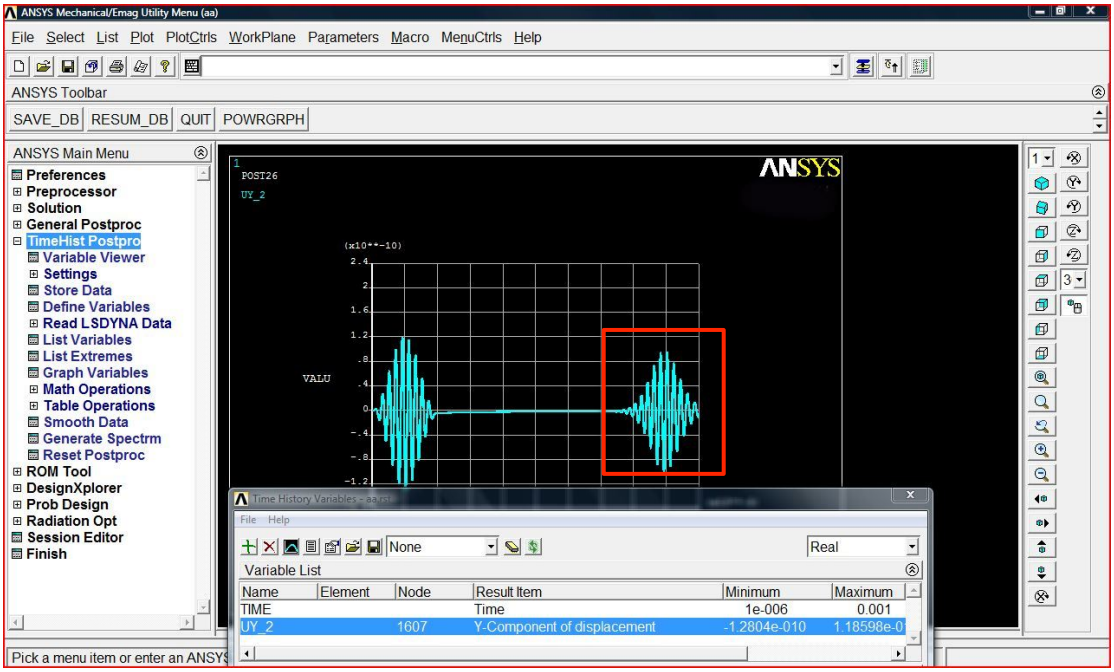


Figure 3.10.3: Y-axis displacement for the signal enforcement node. Rebound existence.

3.11 Rebound correction

With the purpose of checking our hypothesis, it was decided to repeat exactly the same simulation but creating instead a plate long enough (3 meters this time) so that the signal would reach the studied node but being itself unable to arrive at the end of the geometry. This fact would theoretically allow having the nodes 'displacements without any rebound at all. Because of this length change, both surface edges had to be divided in 2400 elements instead of 800, when the plate was 1 meter long. Accordingly, the nodes' number changed; the excitation node became now the 4807 and the node where the propagation was studied became the number 7207. Those were the results attained after changing the plate's length.

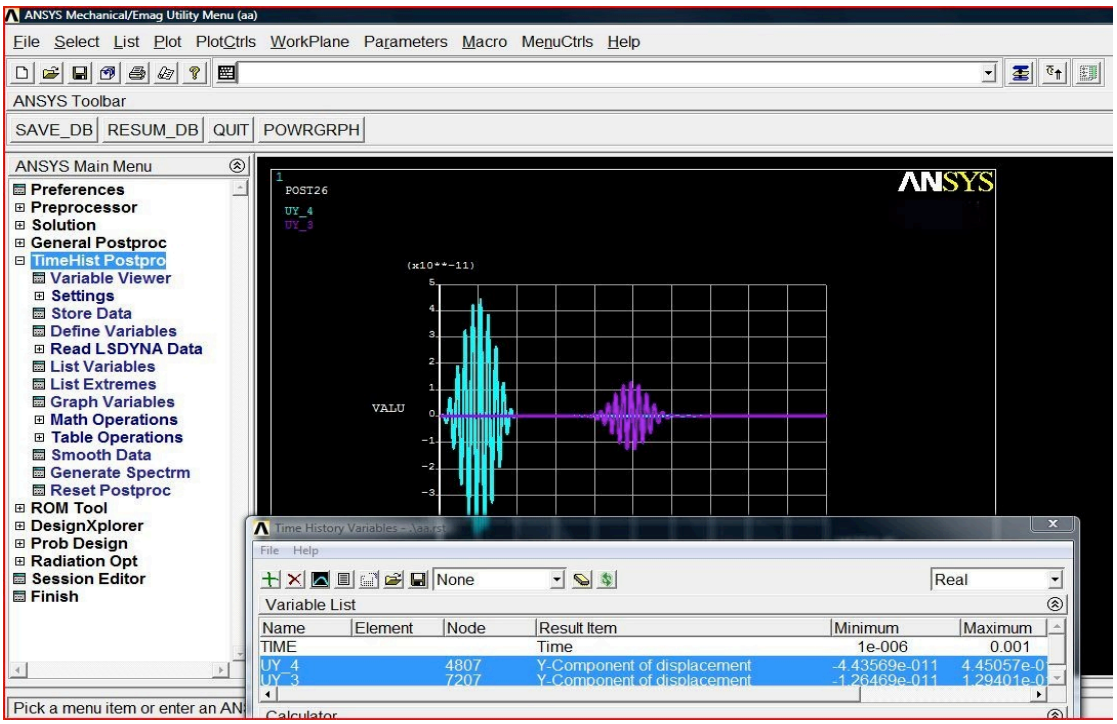


Figure 3.11.1: Results attained of the 3m long simulation. Excitation and Propagation. No existence of rebounds.

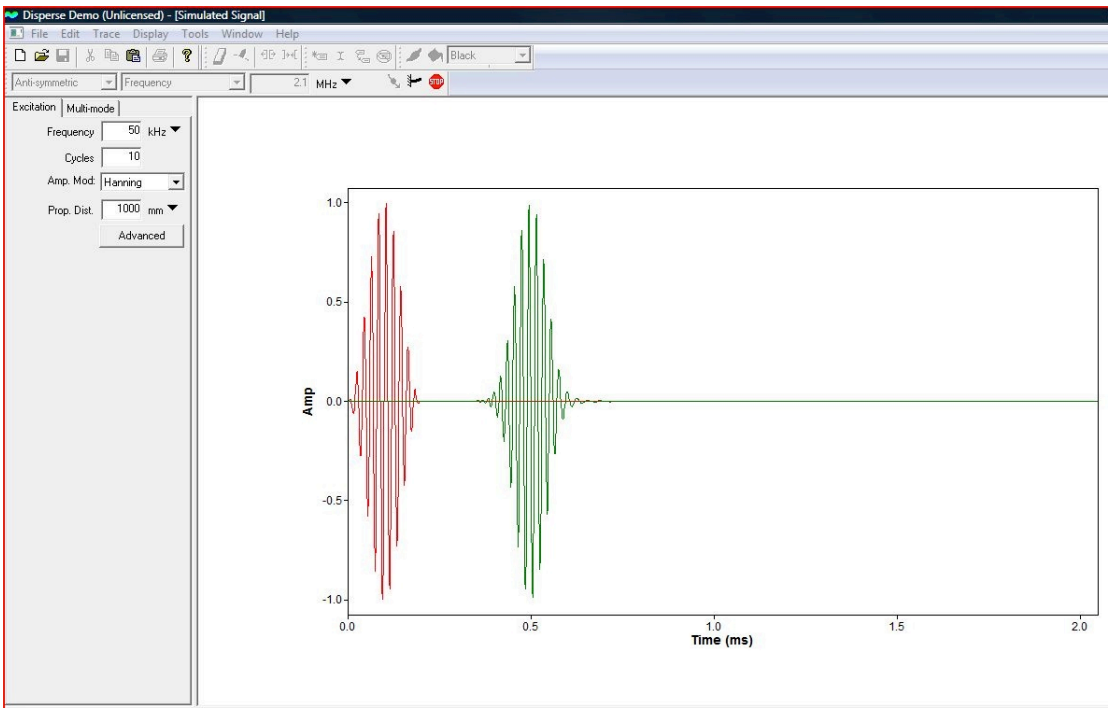


Figure 3.11.2: Reference results made with Disperse.

Even though both software results looked similar, the amplitude of the propagated signal from ANSYS differed from the one from Disperse; the reason of this difference because this last software showed the signals normalized. Nevertheless, in order to ensure that they were entirely exact and the accomplished simulation could be coded, it was required to process them with Matlab and superimpose the results from both ANSYS and Disperse.

When both the excitation and the propagation files from ANSYS were imported, it was notet that both displacement vectors did not have the same length. Its differences were checked and it was seen that in the propagation file, the first 102 values were “Not-a-number” values that needed to be deleted. Surprisingly, at the time to delete them, they appeared instead some incoherent numbers that were substituted by 0 as the propagation displacements vector was being treated. After correcting those internal errors, the vector was then ready to be compared with the reference displacements.

With the purpose of completing that comparison, another script in Matlab was done so as to have all the results in the same graphic in order to check easily if the obtained results were right. Here below is the script described step by step.

```
%% Results Comparison DISPERSE vs. ANSYS (L=3m)
%%
% Disperse results Data Import
disperse=importdata('/Users/lluisortiz/Documents/PFC/Memoria/1.25
3m/Titanium_SimulatedSignals.txt')
%
% Extraction of time vector in Disperse
timedisperse=disperse.data(:,1);
%
% Extraction of excitation displacement vector in Disperse
excitationdisperse=disperse.data(:,2);
%
% Extraction of propagation displacement vector
propagationdisperse=disperse.data(:,4);
%
% ANSYS Excitation results Data Import
excansys=importdata('/Users/lluisortiz/Documents/PFC/Memoria/1.25 3m/4807.csv')
%
% Extraction of time vector in ANSYS
timeansys=excansys.data(:,1);
%
% Extraction of excitation displacement vector in ANSYS
excitationansys=excansys.data(:,2);
%
% ANSYS Propagation results Data Import
propansys=importdata('/Users/lluisortiz/Documents/PFC/Memoria/1.25 3m/7207.csv')
%
% Extraction of propagation displacement vector in ANSYS
propagationansys=propansys.data(:,2);
%
% Correction of Not-a-Number (NaN) values in ANSYS propagation file
propagationansys(isnan(propagationansys))=[];
%
```

```
% Correction of first 102 values in ANSYS propagation file
propagationansys(1:102)=0;
% Normalization of ANSYS excitation displacement vector
normexcitationansys=excitationansys./(max(excitationansys));
%
% Normalization of ANSYS propagation displacement vector
normpropagationansys=propagationansys./(max(propagationansys));
%
% Normalization of Disperse time vector
normtimedisperse=timedisperse./1000;
%
% Excitation Graph Drawing
figure;plot(timeansys,normexcitationansys,normtimedisperse,excitationdisperse);
%
% Propagation Graph Drawing
figure;plot(timeansys,normpropagationansys,normtimedisperse,propagationdisperse;
```

Here are both the excitation and the propagation results compared, obtained after executing the script shown above.

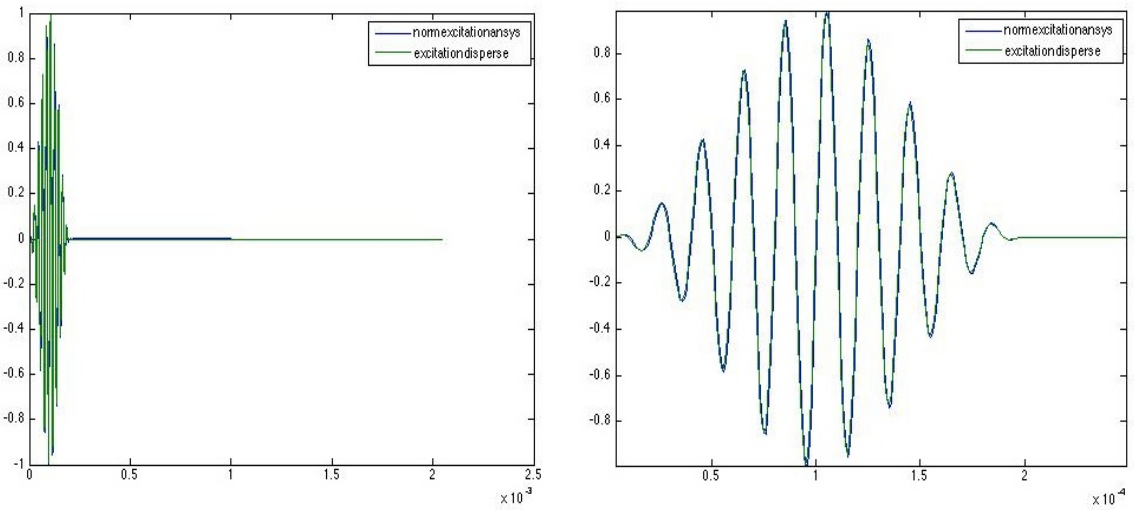


Figure 3.11.3: Excitation results comparison. Superposition of ANSYS and Disperse results' files

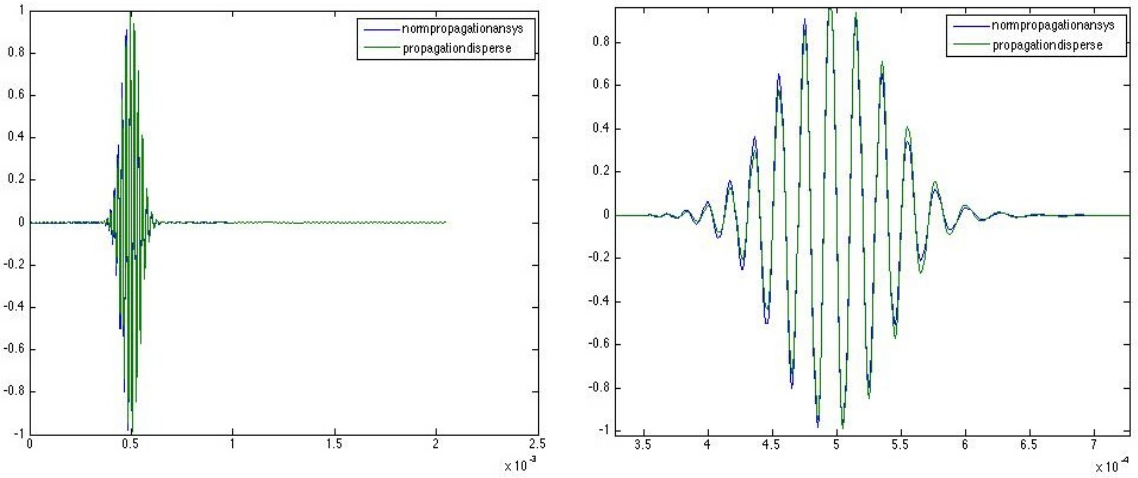


Figure 3.11.4: Propagation results comparison. Superposition of ANSYS and Disperse results' files

The results looked really similar since the graphs from both programs were completely superimposed. That fact signified that the accomplished simulation was correct and accordingly it could be coded with Matlab. Even though the procedure was correct, it needed to be improved because it was useless, both in economic and physic terms, to build a plate three times longer than necessary. However, the fact to shorten the plate could not imply the possible existence of signal rebounds. Thus, a new method to remedy this potential problem needed to be thought.

3.12 The absorbent strips

After some research, a way to solve the existence of signal rebounds was found: adding an absorbent strip at each end of the plate so that they could attenuate completely the signal when it got to each end. Each vertical slice of elements of each absorbent strip needed to incorporate an increasing exponential alpha-damping coefficient from the extreme of the plate until the extreme of our geometry. By having the highest value on the extreme it was ensured that the signal would be attenuated at the end of the plate. Besides, the reason way two absorbent strips were added at each end of the plate instead of one was because in case of having a defect, a rebound would be created at the beginning of the flaw that would also need to be attenuated. Nevertheless, this case would be explained afterwards.

A simple illustration of the plate is shown below, where the darkest regions correspond to the highest alpha-damping coefficient.

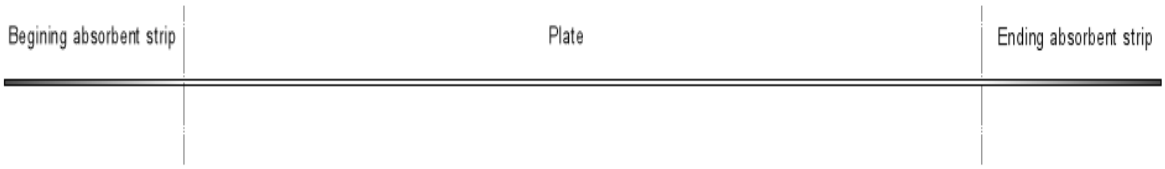


Figure 3.12.1: Plate sketch including the beginning and ending absorbent strips.

The alpha-damping coefficient defined the mass matrix multiplier that formed the viscous damping matrix with the formula [C] = α*[M], where [M] is the mass matrix. The formula for the alpha coefficient was of the form $\boldsymbol{\alpha} = \left(\boldsymbol{K} * \frac{\boldsymbol{i}}{\boldsymbol{Nabs}}\right)^{\boldsymbol{power}}$, where "K" was a constant which defined the maximum value for alpha at the end of the strip (K=1E6), "I" was the element slice that had been treated at that moment, "Nabs" defined the total number of slices of the absorbent strip and finally, "power" corresponded to the exponent to which the K term would be rose.

Hence, the performed steps with the purpose of designing the absorbent strips were at first to know the band's length in order to fulfill their role, then creating as materials as elements' slices were in the strips (assigning to each of them its corresponding alpha-damping coefficient) and finally proving that they attenuated the signal. However this section would be detailed in the next chapter, where the strips were coded.

Chapter 4

MATLAB Codification

4.1 General Parameters. File creation.

Once the simulation was checked and it guaranteed that what had been done was correct, what was done was coding each step it had been fulfilled in the previous simulation. At the heading of our script all the features of the plate were stated so that they could be changed easily if necessary; then all the ANSYS script would be automatically corrected and adapted to the new requests. In other words, the parameters that controlled not only the geometry of the plate (thickness, length, element size, default properties, absorbent strip characteristics) but also the signal's excitation (number of cycles, frequency, excitation time, total simulation time…) needed to be very accessible to the user who was working with the simulation code. It is important to say that all the script parts that they appear with the "%" symbol before them, they would not appear in the input ANSYS file due to it is used to write explanatory notes.

```
% Deletion of all the variables used in previous procedures
clear all
% Deletion of the previous operations in order to have a clear screen
clc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Parameters to define the plate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Geometry of the plate

% Length of the plate [m]
L = 1;
% Thickness of the plate [m]
t=0.005;
%Element length[m]
dx = 0.00125;

%% Default Geometry

% Diameter of the circular default place in the plate's surface
Defect_Diameter = [0.4];
% Depth of the circular default place in the plate's surface
Defect_Depth = [0];
% Position where the plate defaut begins.
Defect_Location = [L/2];

Defect_Diameter = round(Defect_Diameter/dx);
```

```
Defect_Location = round(Defect_Location/dx);


%% Length of the absorbing region and number of elements

% The length of the absorbing region should be between 3 and 5 times larger
than the wavelength of the signal
Labs = 0.225;
% Total length of the plate (beginning strip + plate + end strip)
LT = L+2*Labs;
% Number of slices of elements in the plate
NL = L/dx;
% Total number of element' slices in the plate including both absorbent
strips.
NT = LT/dx;
% Number of elements in the thickness
Nt = t/dx;
% Number of elements in the absorbing boundary
Nabs=Labs/dx;

% Absorbing Region Damping Parameters; ALPHA DAMPING
power = 3; K = 1e6;

%% Excitation parameters

% Total simulation time
T = 0.001;
% Time step
dt = 0.000001;
% Frequency
F = 50e3;
% Number of cycles
nbcycles = 10;
% Excitation time
T1=(nbcycles/F);


%% Material Properties (Titanium)

% Young modulus [Pa]
E = 121127478933;
% Poisson coefficient
poisson = 0.301585731837;
% Density [kg/m3]
density = 4460;
```

When all the parameters that controlled the script were defined and easily accesible for its user, it was proceeded with the writing what would be the ANSYS input file. First of all a method to write the name of the input file incorporating on it the essential simulation characteristics was thought; at the same time it was desirable to keep the name updated as soon as any parameter was changed. As the input file needed to have a ".lgw" extension, it was added at the end of the file name. In order to perform that operation a variable with the file pathname (the directory where the lgw file would be created after executing the script) had to be created.

Then another variable called "filename" was declared where the essential parameters such as the signal frequency, the number of cycles, the length of the

plate and its thickness were included. By checking the file name, it would be easy to identify the values of the variables that had been used to perform the simulation.
In the filename variable as well as in the whole script, the instruction “num2str”, which converted a number into a string variable[1], was used. It is important to remember that all the variables that appear in the script from now on were declared before.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% File name and pathname
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pathname = '/Users/lluisortiz/Documents/PFC/Generate Input file';
filename =
[num2str(F/1e3),'kHz_',num2str(nbcycles),'cycles_L=',num2str(L*1000),'mm_t='
,num2str(t*1000),'mm'];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Writing the *lgw file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fid = fopen([pathname,'/',filename,'.lgw'],'w');
```

Finally, in order to write the input file It was needed to create the variable “fid” by emplying the instruction “fopen”, used in Matlab for opening files. As both variables pathname and filename accompanied the “fopen” command, “fid” opened the indicated file in the corresponding directory. The ending “w” of the instruction indicated that the file needed to be written. In the case that a “r” had been used instead, the “fopen” instruction would have read the file.

4.2 Code’s Heading and Material’s type.

After completing this stage, the coding procedure could be started. Taking the earlier simulation as a reference, it was looked in diverse bibliography for how to translate into code instructions each step previously done with the visual interface. It is important to know that in some cases ANSYS works with a binary choice system: 1 means a selected option while a 0 signifies a discarded choice. In the rest of the cases, the way to select an option is by indicating next to the instruction the number of the option that the simulation requires, which is sorted from 0 onwards.
First of all, it was needed to code the heading of the file, that is to say the directory where ANSYS was installed and the full date when the simulation was executed.

[1] In computer science, a string variable is understood as a data type commonly implemented as an array of words. That stores a sequence of words.

Once finished and as in the previous simulation, the script was started by choosing the structural analysis in the preprocessor settings. After the choice, ANSYS generated automatically a sequence of comments confirming that the Structural analysis had been chosen.

In contrast to the variable declaration those simulation steps needed to be written in the input file. Thus it was needed to use the command “fprintf”, which is used to write formatted data in a text file; this command was used in the whole script given that all the instructions written on it would need to appear in the input file. Besides, it was needed to add the “fid” variable that indicated the file and its pathname where all the instructions would be written.

Furthermore, the instruction “!*\n” was written with the purpose of starting a new line in the input ANSYS file.

```
% HEADING

fprintf(fid , '/BATCH\n');

% Time and date execution.

fprintf(fid , '! /COM,ANSYS RELEASE 14.0    UP20111024       13:03:01
03/06/2014\n');

% Start of ANSYS 14.0

fprintf(fid , '/input,start140,ans, ');

% Directory where ANSYS was installed.

fprintf(fid , '''C:\\Program Files\\ANSYS
Inc\\v140\\ANSYS\\apdl\\'',,,,,,,,,,,,,,,1\n');
fprintf(fid, '!*\n');

% PREPROCESSOR

fprintf(fid, '/NOPR\n');
fprintf(fid, 'KEYW,PR_SET,1\n');

% Choice of structural analysis.

fprintf(fid, 'KEYW,PR_STRUC,1\n');

% Discard of Thermal, Fluid, Electromagnetic analysis.

fprintf(fid, 'KEYW,PR_THERM,0\n');
fprintf(fid, 'KEYW,PR_FLUID,0\n');
fprintf(fid, 'KEYW,PR_ELMAG,0\n');
fprintf(fid, 'KEYW,MAGNOD,0\n');
fprintf(fid, 'KEYW,MAGEDG,0\n');
fprintf(fid, 'KEYW,MAGHFE,0\n');
fprintf(fid, 'KEYW,MAGELC,0\n');
fprintf(fid, 'KEYW,PR_MULTI,0\n');

% COMMENTS

fprintf(fid, '/GO\n');
fprintf(fid, '!*\n');
fprintf(fid, '! /COM,\n');
```

```
fprintf(fid, '! /COM,Preferences for GUI filtering have been set to
display:\n');
fprintf(fid, '! /COM,  Structural\n');
```

What followed the previous step was choosing the material type. Before coding this instruction, the command "/prep7" needed to be added in order to enter the general input data preprocessor. Then, as it was explained before, the best material type for our plate was the "plane182" due to its UX/UY degrees of freedom and because the element input data included 4 nodes.

```
% PREPROCESSOR - MATERIAL TYPE

fprintf(fid, '!*\n');


% General Input data preprocessor command

fprintf(fid, '/PREP7\n');
fprintf(fid, '!*\n');

% Quad4node182 material choice.

fprintf(fid, 'ET,1,PLANE182\n');
fprintf(fid, '!*\n');
fprintf(fid, 'KEYOPT,1,1,0\n');
fprintf(fid, 'KEYOPT,1,3,2\n');
fprintf(fid, 'KEYOPT,1,6,0\n');
```

4.3 Materials' generation.

Once the material type was chosen, it was high time to introduce the physical properties of the material the plate would be made of. As it was mentioned before, two absorbent strips had to be created, each of them with an increasing exponential damping coefficient for each element slice. Thus, it was precise to create as different materials as number of element slices was the absorbent strip composed of, besides the plate's material which lacked damping coefficient (ALPD = 0). Consequently, a loop was defined so as to create as materials as element slices. The Young modulus (EX), the Poisson coefficient (PRXY) and the material density (DENS) were constant, thus the loop just affected the damping value (ALPD), that increased as the element slice was closer to the end of the absorbent strip.

Each different material was coded with a different number from one to the number of slices, plus the material of the plate; this material id needed also to be included in the loop. Otherwise, the materials would have been always overwritten. This instruction is introduced by the instruction " MPDATA, *PROPERTY*, num2str(1) ", were the number between parenthesis indicated the material id, in this case the material of the

plate. In the script appears the command “MPTEMP,1,0” and means that all the physical material properties do not correspond to any particular temperature.

```
% MATERIAL DEFINITION

% MATERIAL PLATE 1M

fprintf(fid, '!*\n');
fprintf(fid, 'MPTEMP,,,,,,,,\n');
fprintf(fid, 'MPTEMP,1,0\n');

% Young Modulus for the plate material

fprintf(fid, ['MPDATA,EX,',num2str(1),',',',',num2str(E),'\n']);

% Poisson coefficient for the plate material.

fprintf(fid, ['MPDATA,PRXY,',num2str(1),',',',',num2str(poisson),'\n']);
fprintf(fid, 'MPTEMP,,,,,,,,\n');
fprintf(fid, 'MPTEMP,1,0\n');

% Titanium density.

fprintf(fid, ['MPDATA,DENS,',num2str(1),',',',',num2str(density),'\n']);
fprintf(fid, 'MPTEMP,,,,,,,,\n');
fprintf(fid, 'MPTEMP,1,0\n');

% Damping Coefficient.

fprintf(fid, ['MPDATA,ALPD,',num2str(1),',',',',num2str(0),'\n']);
```

When the plate material was coded and it verified that it worked correctly, it was proceeded with the coding of the absorbing materials loop. It must be taken into account that the only terms that were affected by the loop were the materials’ id and the alpha-damping coefficient, both of them dependents of the “I” variable. In this section, the material id appeared with the instruction “ MPDATA, *PROPERTY*, num2str(i+1) ”, with the purpose of creating at each loop step a new material. Furthermore, the alpha-damping coefficient was also included in the loop with the purpose of creating a new material as far as the “I” variable increased.

```
% Absorbing strip materials

fprintf(fid, '!*\n');

% Loop Start (Nabs materials)
for i=1:Nabs

% Alpha Damping coefficient Definition.
alpha=K*(i/Nabs)^power;

fprintf(fid, 'MPTEMP,,,,,,,,\n');
fprintf(fid, 'MPTEMP,1,0\n');

% Young Modulus for each absorbent strip element slice

fprintf(fid, ['MPDATA,EX,',num2str(i+1),',',',',num2str(E),'\n']);
```

```
% Poisson coefficient for each absorbent strip element slice

fprintf(fid, ['MPDATA,PRXY,',num2str(i+1),',',',',num2str(poisson),'\n']);
fprintf(fid, 'MPTEMP,,,,,,,,\n');
fprintf(fid, 'MPTEMP,1,0\n');


% Titanium density for each absorbent strip element slice

fprintf(fid, ['MPDATA,DENS,',num2str(i+1),',',',',num2str(density),'\n']);
fprintf(fid, 'MPTEMP,,,,,,,,\n');
fprintf(fid, 'MPTEMP,1,0\n');

% Alpha Damping coefficient for each absorbent strip element slice

fprintf(fid, ['MPDATA,ALPD,',num2str(i+1),',',',',num2str(alpha),'\n']);

end
fprintf(fid, '!*\n');
```

4.4 Meshing Procedure and Nodes' Allocation.

Arrived at this point, it was necessary to think how the geometry of the plate would be designed. To that end, it was essential at first to know how to distribute the nodes that would form the mesh of the plate; that meant that it was crucial to understand how to number each node of the plate. It was required a numbering system that allowed the modeling to maintain the same node number despite any kind of changes. Obviously if the element size changed, the total number of nodes would change accordingly but what was vital was to always maintain the same numbering pattern.

Besides, it was contemplated the possibility to introduce in our geometry a damage spot in the plate's surface. This new item caused some changes like the must of introducing two absorbing strips instead of one or the different element generation process, as it would be shown later on. The signal was implemented in the left extreme of the plate following the y-axis direction, propagating itself along the plate section from left to right. At the time the signal hit an obstacle like the edge where the damage spot began, a rebound of the signal was created, forcing us to add another absorbing strip where the signal was applied. If the spot was not placed in the surface of the plate, the left absorbing strip would not be necessary because any rebound would be created; just with the right strip, the whole signal would be lessened. The reason why this flaw was inserted in the surface of the plate was with the purpose of simulating a piece of a plate whose thickness had been damaged, making that the script looked like a real pipeline's thickness study.

The mesh of the plate required being a grid, what meant that the nodes would need to be distributed in both directions x-axis and y-axis. Thus, it was thought that numbering the nodes with high number could be useful for that numbering pattern. We imposed the node of the lower left corner of the plate as the number 10001. That node number would not change on no account because it would be our reference node for meshing the geometry.

The way how the nodes 'id changed needed to be different according to the axis direction: the number increase between two consecutive nodes in the y-axis direction was 10000 while in the x-axis direction was just 1. The reason of this criterion was because the number of elements along the thickness of the plate would always be smaller than the number of elements along its length.

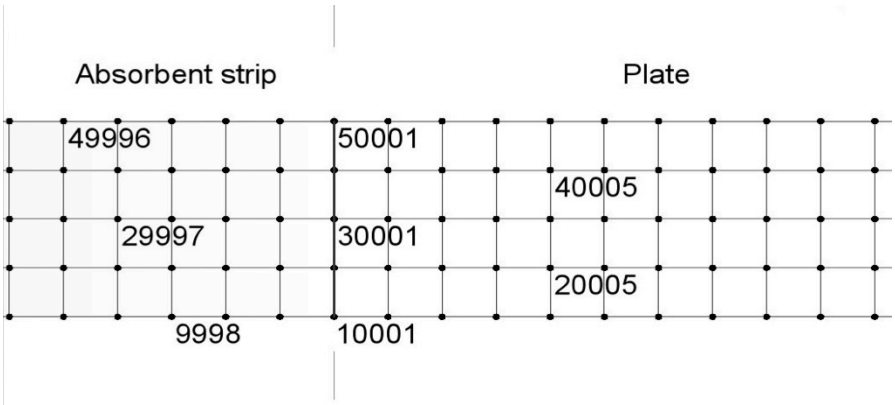


Figure 4.4.1. Mesh fragment. Numbering pattern.

This way, it was easy to locate and identify the key nodes of our geometry, that were placed on the lateral edges of the plate (beginning and ending) as well as the nodes that belonged to both absorbent strips. Furthermore, this numbering system became essential in order to create correctly the elements of the plate section and linking them accordingly to their corresponding material.

The reason why this numbering system was used was because it would be very rare to have more than 10000 nodes on a surface edge; if that was the case, the nodes' id increment in the y-axis direction would became useless as the increase of two nodes placed one above the other was 10000. It must be taken into account that this simulation was done with the purpose of comparing two software solutions, what meant that it was not necessary to simulate the plate with such a precise methodology.

In order to perform this node grid, the ANSYS commands N (Node) and NGEN (Node Generator) were employed. With regard to the command N, the input data

structure used was the following one: “N, NODE (zero or negative), X, Y, Z, THXY, THYZ, THZX”, where “NODE” referred to the node number or id, “X,Y,Z” meant the coordinates where the node would be placed and “THXY, THYZ, THZX” were the rotations about the three axis.

```
% Beginning absorbent strip nodes
% Location of the first node (lower left corner of the beginning strip)

fprintf(fid, ['N,',num2str(10001-Nabs),',',num2str(0),',0,0,0,0,0\n']);
```

This instruction meant that the node which number was the “10001-Nabs” was located in the coordinates (0,0,0) and had no rotation in any axis direction.

Referring to the NGEN command, the input data structure used was “NGEN, ITIME, INC, NODE1, NODE2, NINC, DX, DY, DZ, SPACE”, where “ITIME” meant the total times this generation operation would be done, “INC” was the numeric increment between one node and its immediately posterior, “NODE1” was the node where this generation started while “NODE2” was the node where the procedure ended. In this case, there was not any ending node, so what was done was making NODE1 and NODE2 equal and consequently imposing NINC equal to 1. Finally “DX, DY DZ” were the distance increments in the three axis directions between a node and its posterior. Finally, “SPACE” was equal to 1 by default.

```
% Node generation for each node layer in the left edge of the beginning
absorbent strip .

fprintf(fid, ['NGEN,',num2str(Nt+1),',10000,',num2str(10001-
Nabs),',',num2str(10001-Nabs),',1,0,',num2str(dx),',0,1\n']);
```

This instruction what meant was that “Nt+1” nodes (as nodes as elements in the thickness plus the node of the inferior surface edge) were being generated, where the numeric increment between a node and its following was 10000, which starting generation node was the one created previously, and whose distance increment between two consecutive nodes was the elements size was 0 in the x-axis direction, the element size (dx) in the y-axis direction and 0 in the z-axis direction. In other words, all the nodes of the left lateral absorbent strip edge were being built from which it would be possible to generate the rest of the nodes.

```
% Loop node generator for the rest of the strip.

for i=0:(Nt)
    fprintf(fid, ['NGEN,',num2str(Nabs+1),',',num2str(1),',',num2str(10001-
Nabs+i*10000),',', num2str(10001-
Nabs+i*10000),',',num2str(1),',',num2str(dx),',',num2str(0),',',num2str(0),'
```

```
,',num2str(1),'\n']);
end
```

When the lateral edge nodes were built, it was required to create the rest of the nodes for the plate. To perform this stage, a loop was created again so as to create as levels of nodes as elements were in the thickness. The instruction used was very similar to the one used before despite the fact that the variables make it seem more complicated.

What this instruction said was that we were generating “Nabs+1” nodes (the number of slices of elements in the strip plus the node that had been created before) for “Nt+1” times (the number of elements in the thickness plus the inferior layer). This generation procedure had as initial and ending node the created ones in the previous step, always corresponding to the level of nodes it had been generated. The instruction “ […] num2str(10001-Nabs+i*10000) […] ” was used in order to jump from one level to another when the “NGEN” command finished generating all the nodes for that horizontal. Besides, the distance increment between two consecutive nodes was again the size of the element in the x-axis directions while the distance increment in the rest of the directions was null.

Once the left absorbing strip’s nodes were generated, the next step was building the nodes that belonged to the plate. With the purpose of performing this task, the same method as before was repeated, creating another loop but this time generating “NL+1” nodes (the number of slices of elements in the plate plus the created one) instead of “Nabs+1”. Besides, the nodes located at the end of the strip were taken as starting and ending ones.

```
% Nodes plate

for i=0:(Nt)
    fprintf(fid,
['NGEN,',num2str(NL+1),',',num2str(1),',',num2str(10001+i*10000),',',
num2str(10001+i*10000),',',num2str(1),',',num2str(dx),',',num2str(0),',',num
2str(0),',',num2str(1),'\n']);
end
fprintf(fid, '!*\n');
```

As before, the loop was repeated one last time in order to build the nodes that belonged to the ending absorbent strip; what changed now was the starting and ending node. Instead of generating the level of nodes from the ones located in the left lateral edge of the strip, the procedure began from the nodes placed at the right end side of the plate. That is why the instruction “num2str(10001+NL+i*10000)” was used.

```
% Nodes Bande Absorbante Fin
for i=0:(Nt)
    fprintf(fid,
['NGEN,',num2str(Nabs+1),',',num2str(1),',',num2str(10001+NL+i*10000),',',
num2str(10001+NL+i*10000),',',num2str(1),',',num2str(dx),',',num2str(0),',',
num2str(0),',',num2str(1),'\n']);
end
fprintf(fid, '!*\n');
```

It was mentioned at the beginning the possibility to add a default on the plate's surface. It would seem understandable to think about the reason why it had designed the complete grid of nodes if a hole existed. The reason why the whole grid of nodes was built was because they did not disturb in the simulation process and at the same time they allowed the user to generate, if desired, the plate without any default at all.

4.5 Elements' generation.

It was the moment to generate the elements. This was a tedious procedure because the place where the flaw was located had to be parameterized as well, keeping always the possibility of modelling the plate without any hole in its surface. The way to generate the elements was very similar to the method used to create the nodes. At first the command "E" was employed with the purpose of generating the first element of the plate; the input data structure of the E command was *"E, I, J, K, L, M, N, O, P",* where the letters I to K were the nodes' id in anti-clockwise order that were needed to build an element. Besides, the command "EGEN" was employed too, whose use was to generate elements in the same way the nodes were previously placed. Its input data structure was "EGEN, ITIME, NINC, IEL1, IEL2, IEINC, MINC, TINC, RINC, CINC, SINC", where ITIME was the total times the generation operation would be executed, "NINC" was the pattern in which all the nodes incremented, "IEL1" and "IEL2" were respectively the starting and ending elements between which the generation operation would be done, "IEINC" were the steps done to achieve the generation task, "MINC" was the material number increment between an element and its immediately posterior, "TINC" was the type increment between two consecutive elements and finally, "CINC" and "SINC" were respectively the coordinate system and section ID number increments.

This element generation was divided in different steps. At first what was done was generating all the nodes of the plate, from the left edge of the plate to the right one, that were under the default. An explanatory example: if the hole had a depth of 0.0025 m and the element size was 0.00125 m, the elements that would have been created were the first 2 levels. Here is an image to show it.

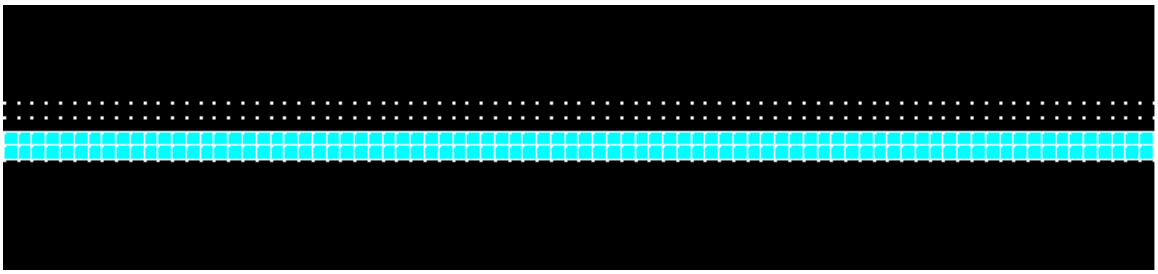


Figure 4.5.1: Elements creation. 2 first levels created.

```
for i=0:(Nt-(Defect_Depth/dx)-1)
fprintf(fid,
['E,',num2str(10001+i*10000),',',num2str(10002+i*10000),',',num2str(20002+i*
10000),',',num2str(20001+i*10000),'\n']);
fprintf(fid,
['EGEN,',num2str(NL),',',num2str(1),',',num2str(1+(i*NL)),',',num2str(1+(i*N
L)),',',num2str(0),',',num2str(0),'\n']);
end
fprintf(fid, '!*\n');
```

What this loop means is that the element generation would be done as time as elements in the thickness were below the default. The “E” command was used to generate the elements of the plate located in the left edge, where the reference node (10001) was placed. The instruction “…+i*1000” was used to rise up to the following elements’ levels as the loop finished generating a layer of elements. By default, ANSYS attributed the id number 1 to the element recently created. Thus, the elements ‘ids of a layer would be from 1 to 800, from 801 to 1600 etc.

What it was pretended with the “EGEN” command was to repeat the generation as times as elements were in a layer of the plate. The instruction said that the operation would take place with a node increment equal to 1, and that the starting and ending element was the one created in the left edge of the plate. That is the reason why the instruction “num2str(1+(i*NL))” was used, just in order to take always the reference element independently of its layer. As the only plate needed to have the same material, a null material increment between the elements’ slices was set. In the “EGEN” input data structure, some items were missing; they were omitted because it was useless to use them. ANSYS allowed deleting the items from a command if they were useless for a simulation.

Then, once the base of the plate was built, it was proceeded with the creation of the elements of the rest of the plate’s levels from the left side until the beginning of the hole. Here below is an illustration so as to see it clearly.

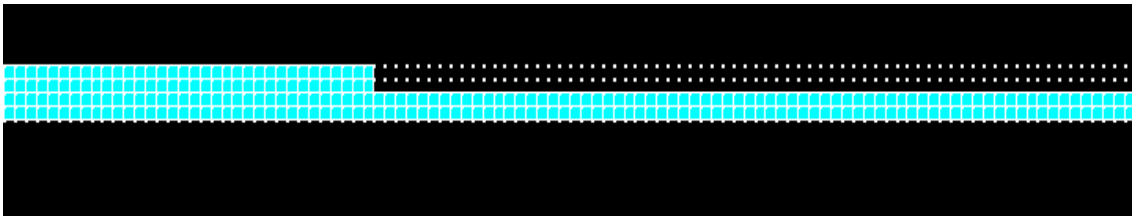


Figure 4.5.2: Elements creation. Beginning of the upper levels elements.

```
for i=0:((Defect_Depth/dx)-1)
fprintf(fid, ['E,',num2str(10001+i*10000+(Nt-
(Defect_Depth/dx))*10000),',',num2str(10002+i*10000+(Nt-
(Defect_Depth/dx))*10000),',',num2str(20002+i*10000+(Nt-
(Defect_Depth/dx))*10000),',',num2str(20001+i*10000+(Nt-
(Defect_Depth/dx))*10000),'\n']);
fprintf(fid,
['EGEN,',num2str(Defect_Location),',',num2str(1),',',num2str(1+((Nt-
(Defect_Depth/dx))*NL)+(i*Defect_Location)),',',num2str(1+((Nt-
(Defect_Depth/dx))*NL)+(i*Defect_Location)),',',num2str(0),',',num2str(0),'\
n']);
end
fprintf(fid, '!*\n');
```

The function of this loop was generating as layers of elements as deep was the default located in the plate's surface. The generation operation started again in the left edge of the plate but this time it finished where the hole began. To perform this task, the reference element was built again in the left edge of the plate, although it needed to be located at the same level than the deepest one of the hole. What was done afterwards was generating as nodes as fitted between the left end edge of the plate and the beginning of the hole; to achieve it, we used the instruction "num2str(Defect_Location)", which transformed the distance where the default began into the number of nodes that fitted between the beginning of the plate and the hole. This time the way to refer to the starting and ending element used was "num2str(1+((Nt-(Defect_Depth/dx))*NL) + (i*Defect_Location))"; this instruction indicated the elements placed in the left end edge of the plate that were on the same layer than the hole.

Finally, what was done was creating as well the nodes of the rest of the plate's levels but this time from the end of the default until the end of the plate.

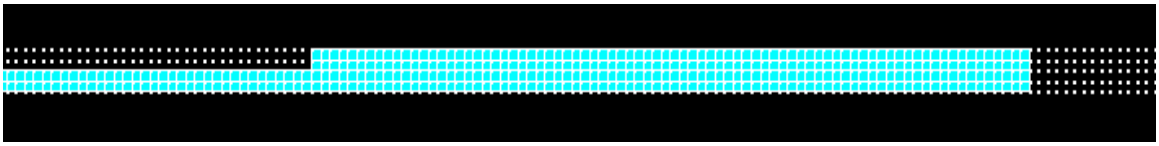


Figure 4.5.3: Elements creation. Ending of the upper levels elements

```
for i=0:((Defect_Depth/dx)-1)
fprintf(fid, ['E,',num2str(10001+i*10000+(Nt-
(Defect_Depth/dx))*10000+Defect_Diameter+Defect_Location),',',num2str(10002+
i*10000+(Nt-
(Defect_Depth/dx))*10000+Defect_Diameter+Defect_Location),',',num2str(20002+
i*10000+(Nt-
(Defect_Depth/dx))*10000+Defect_Diameter+Defect_Location),',',num2str(20001+
i*10000+(Nt-
(Defect_Depth/dx))*10000+Defect_Diameter+Defect_Location),'\n']);
fprintf(fid, ['EGEN,',num2str(NL-Defect_Location-
Defect_Diameter),',',num2str(1),',',num2str(1+((Nt-
(Defect_Depth/dx))*NL)+((Defect_Depth/dx)*Defect_Location)+(i*(NL-
Defect_Location-Defect_Diameter))),',',num2str(1+((Nt-
(Defect_Depth/dx))*NL)+((Defect_Depth/dx)*Defect_Location)+(i*(NL-
Defect_Location-Defect_Diameter))),',',num2str(0),',',num2str(0),'\n']);
end
fprintf(fid, '!*\n');
```

This loop was done following the same method but this time what changed was that the reference element was placed in the edge where the hole finished. Then, what was done was filling with elements from the right end edge of the hole to the right end edge of the plate. That is why this generation operation was done “NL-Defect_Location-Defect_Diameter” times; the way to indicate the starting and the ending element was “num2str(1 + ((Nt - (Defect_Depth/dx)) * NL) + ((Defect_Depth/dx) * Defect_Location) + (i*(NL-Defect_Location-Defect_Diameter)))”.

Once the plate was designed, including the parameterization of the hole’s sizing and its location, it was proceeded with the elements generation coding for the ending and beginning absorbent strips respectively. In order to succeed, it was needed to use the command “EMODIF”, which was used to modify the properties of a previously defined element. The command followed the input data structure “EMODIF, IEL, STLOC, I1, I2, I3, I4, I5, I6, I7, I8”, where “IEL” indicated the element number or id it is required to modify, “STLOC” meant either the starting location of the node to be modified or different properties such as the material (MAT) or the type of element (TYPE) and “I1”, in case of choosing MAT or TYPE, the value that replaced the attributed before. The reason why we used this command was because ANSYS assigned by default the material number 1 (the material without any damping value)

to any created element; in order to set the properties of the absorbent strips, we had to change the material id to some nodes. This would be explained afterwards. Here below is explained how the ending absorbent strip was coded.

```
% Ending absorbent strip elements

for j=0:(Nt-1)
fprintf(fid,
['E,',num2str(10001+NL+j*10000),',',num2str(10002+NL+j*10000),',',num2str(20
002+NL+j*10000),',',num2str(20001+NL+j*10000),'\n']);
fprintf(fid, ['EMODIF,',num2str(1+((Nt-
(Defect_Depth/dx))*NL)+((Defect_Depth/dx)*Defect_Location)+((Defect_Depth/dx
)*(NL-Defect_Location-Defect_Diameter))+(j*Nabs)),',MAT,',num2str(2),'\n']);
fprintf(fid, ['EGEN,',num2str(Nabs),',',num2str(1),',',num2str(1+((Nt-
(Defect_Depth/dx))*NL)+((Defect_Depth/dx)*Defect_Location)+((Defect_Depth/dx
)*(NL-Defect_Location-Defect_Diameter))+(j*Nabs)),',',num2str(1+((Nt-
(Defect_Depth/dx))*NL)+((Defect_Depth/dx)*Defect_Location)+((Defect_Depth/dx
)*(NL-Defect_Location-
Defect_Diameter))+(j*Nabs)),',',num2str(0),',',num2str(1),'\n']);
fprintf(fid, '!*\n');
end
fprintf(fid, '!*\n')
```

With this loop, as layers as total elements were in the thickness were generated. As it was done previously, the reference element was placed where the ending absorbent strip began, in the right ending edge of the plate. It was explained before that the slice of elements nearer to the plate needed to have the lowest damping value, whose corresponding material was number 2. Consequently, it was required to change its material id from 1 to 2, which was done with the “EMODIF” command.
Furthermore, the “EGEN” command was used in order to generate all the elements that composed the strip. With this instruction, it existed the possibility to increase the material id between 2 consecutive elements, a tool that was used so as, finally, to assign the highest alpha-damping value to the farthest strip’s slice of the plate.
Here below is the accomplished script.

```
% Beginning absorbent strip elements

for j=0:(Nt-1)
fprintf(fid, ['E,',num2str(10001-Nabs+j*10000),',',num2str(10002-
Nabs+j*10000),',',num2str(20002-Nabs+j*10000),',',num2str(20001-
Nabs+j*10000),'\n']);
fprintf(fid, ['EMODIF,',num2str(1+((Nt-
(Defect_Depth/dx))*NL)+((Defect_Depth/dx)*Defect_Location)+((Defect_Depth/dx
)*(NL-Defect_Location-
Defect_Diameter))+(Nt*Nabs)+(j*Nabs)),',MAT,',num2str(Nabs+1),'\n']);
fprintf(fid, ['EGEN,',num2str(Nabs),',',num2str(1),',',num2str(1+((Nt-
(Defect_Depth/dx))*NL)+((Defect_Depth/dx)*Defect_Location)+((Defect_Depth/dx
)*(NL-Defect_Location-
Defect_Diameter))+(Nt*Nabs)+(j*Nabs)),',',num2str(1+((Nt-
(Defect_Depth/dx))*NL)+((Defect_Depth/dx)*Defect_Location)+((Defect_Depth/dx
)*(NL-Defect_Location-
Defect_Diameter))+(Nt*Nabs)+(j*Nabs)),',',num2str(0),',',num2str(-1),'\n']);
fprintf(fid, '!*\n');
end
```

```
fprintf(fid, '!*\n');
```

For the beginning absorbent strip, the procedure was exactly the same as the ending one but this time, the reference element was placed in the left edge of the strip, so that the elements were created from left to right until arriving to the plate. By default, ANSYS assigned to that element the material id number 1 but it needed to have the number “Nabs+1” (the material with the highest damping value) so that the strip worked correctly. With the “EGEN” command, the elements were built up to the plate but this time decreasing the material id one unity between one element and its immediately posterior; the loop was used in order to create the different elements’ layers so that the strip could be built following the decreasing element slices’ damping values as mentioned previously.

4.6 Transient Analysis and Node’s Fixation.

After concluding the modeling and meshing procedures, the coding procedure continued with the creation of the transient analysis that would allow to perform the simulation later on. With the intention of achieving the goal, the command “ANTYPE” (Analysis Type), whose input data structure was “ANTYPE, Antype, Status”, was used. ANSYS offered different kind of analysis such as Static, Buckle (to perform buckling analysis), Modal, Harmonic (HARMIC) or Transient (TRANS) amongst others. In order to select the Transient mode, we needed to give ANSYS the instruction to select the 4th type (Static mode was number 0). So as to chose a full mode as explained in the previous simulation, the command “TRNOPT” (Transient Option Analysis) was employed. Finally, the lumped mass matrix needed to be disabled, that is why we added a 0 next to the “LUMPM” command. Here below is the script executed.

```
% SET TRANSIENT ANALYSIS

fprintf(fid, 'ANTYPE,4\n');
fprintf(fid, 'TRNOPT,FULL\n');
fprintf(fid, 'LUMPM,0\n');
fprintf(fid, '!*\n');
```

The next stage was then fixing a node to avoid having a non-singular stiffness matrix. The fix node needed to be placed in one corner of the geometry so as to avoid having wrong results; this time the node was placed instead in the right end edge of the ending absorbent strip. The reason why the location of the node was changed

was because of the beginning strip; if the node was fixed in the left side of the geometry, the results obtained were not as precise as it was required.

```
% APPLY DISPLACEMENT NODE

fprintf(fid, ['D,',num2str(10001+Nabs+NL+(10000*Nt)),',ALL\n']);
fprintf(fid, '!*\n');
```

4.7 Functions and windows' codification and Enforcement.

As it was mentioned in the introduction of the report, a script as complete as possible was wanted. That is why instead of coding just a Hanning window, it was decided to code the most important ones and let the user of the script choose the window he/she needed. It was taken into consideration that the most important windows for Lamb waves were Hanning, Hamming and Gauss. In the script, the instruction "Switch" was used so that the user could easily select the required window. This command allowed to include each window in a different case so that when a type of function was chosen, the rest became disabled; the different cases were closed with the "end" instruction. Those were the different windows coded:

$$Hanning: f(t) = \left(0.5 * \left(1 - \cos\frac{2 * \pi * t}{0.0002}\right)\right) * \sin(2 * \pi * 50000 * t)$$

$$Hamming: f(t) = \left(0.53836 - \left(0.46164 * \cos\frac{2 * \pi * t}{0.0002}\right)\right) * \sin(2 * \pi * 50000 * t)$$

$$Gauss: f(t) = e^{-\frac{1}{2} * \left(\frac{\left(t - \frac{0.0002}{2}\right)}{0.275 * \frac{0.0002}{2}}\right)^2}$$

ANSYS' method to state equations was very particular since they were coded by parts; in other words, each piece of the equations was saved in a Temporary Register Variable, that was combined at the same time with another Register Variable (that stores another piece of the equation) to complete the whole function. Furthermore, not only the mathematical operations but also the variables (such as TIME, X, Y, Z. DENS…) were stored in a table that was necessary to employ.

Operation Number	Operation
1	ADD
2	SUBTRACT
3	MULTIPLY
4	DIVIDE
5	LOG
6	LOG10
7	POWERE
8	POWER10
9	SIN
10	COS
11	TAN
12	ASIN
13	ACOS
14	ATAN
15	ABS
16	SQRT
17	POWER
18	ATAN2
19	MIN
20	MAX
21	INVPOWER

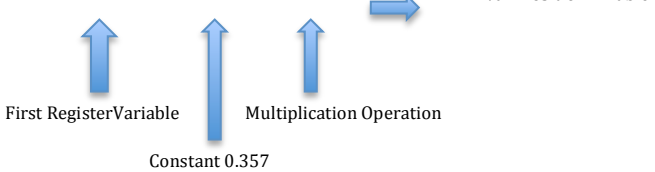
Figure 4.5. Table of Mathematical Operations

Variable Number	Primary Variable
1	ParTabNames(1)(1:8) = 'TIME '
2	ParTabNames(2)(1:8) = 'X '
3	ParTabNames(3)(1:8) = 'Y '
4	ParTabNames(4)(1:8) = 'Z '
5	ParTabNames(5)(1:8) = 'TEMP '
6	ParTabNames(6)(1:8) = 'VELOCITY'
7	ParTabNames(7)(1:8) = 'PRESSURE'
8	ParTabNames(8)(1:8) = 'TSURF '
9	ParTabNames(9)(1:8) = 'DENS '
10	ParTabNames(10)(1:8)= 'SPHT '
11	ParTabNames(11)(1:8)= 'KXX '
12	ParTabNames(12)(1:8)= 'KYY '
13	ParTabNames(13)(1:8)= 'KZZ '
14	ParTabNames(14)(1:8)= 'VISC '
15	ParTabNames(15)(1:8)= 'EMIS '
16	ParTabNames(16)(1:8)= 'MAT '
17	ParTabNames(17)(1:8)= 'CONST1 '
18	ParTabNames(18)(1:8)= 'CONST2 '
19	ParTabNames(19)(1:8)= 'CONST3 '
20	ParTabNames(20)(1:8)= 'CONST4 '
21	ParTabNames(21)(1:8)= 'CONST5 '
22	ParTabNames(22)(1:8)= 'CONST6 '
23	ParTabNames(23)(1:8)= 'CONST7 '
24	ParTabNames(24)(1:8)= 'CONST8 '
25	ParTabNames(25)(1:8)= 'CONST9 '
26	ParTabNames(26)(1:8)= 'SECTOR '
27	ParTabNames(27)(1:8)= 'Xr '
28	ParTabNames(28)(1:8)= 'Yr '
29	ParTabNames(29)(1:8)= 'Zr '
30	ParTabNames(30)(1:8)= 'GAP '
31	ParTabNames(31)(1:8)= 'OMEGS '
32	ParTabNames(32)(1:8)= 'OMEGF '
33	ParTabNames(33)(1:8)= 'SLIP '

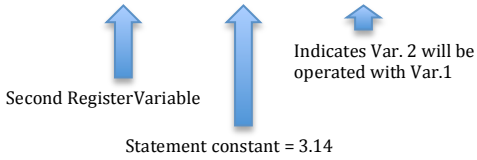
Figure 4.6. Table of variables

Here are some examples about how the data in the function builder was written.

- “*SET,%_FNCNAME%(0,1,1), 1.0, -1, 0, 1, 0.357, 3, 1”



- “*SET,%_FNCNAME%(0,1,1), 0.0, -2, 0, 3.14, 0, 0, -1”



It is important to state the difference between working with a Register Variable or with a variable that appears in the table shown before. The way to distinguish them is because the Variable Register carries a "-" sign while the variable tabulated did not need to carry it. Needless to say that the structure shown in the examples above was identical to the one followed for developing all the functions. Here is shown how all the window equations were coded.

```
% Function Application

window='hanning';

switch (window)

    case 'hanning'

fprintf(fid, '*DEL,_FNCNAME\n');
fprintf(fid, '*DEL,_FNCMTID\n');
fprintf(fid, '*DEL,_FNCCSYS\n');
% Name of the Window
fprintf(fid, '*SET,_FNCNAME,''HANNING''\n');
fprintf(fid, '*SET,_FNCCSYS,0\n');
% Importation of the .func file where the information of the function is
stored
fprintf(fid, '! /INPUT,.\\functions\\hann.func,,,1\n');
% Table of values of 20 operations, 3 regimes
fprintf(fid, '*DIM,%%_FNCNAME%%,TABLE,6,20,3,,,,%%_FNCCSYS%%\n');
fprintf(fid, '!\n');
% Statement of the TIME variable
fprintf(fid, '! Begin of equation: {TIME}\n');
% -999 represents function in table format
fprintf(fid, '*SET,%%_FNCNAME%%(0,0,1), 0.0, -999\n');
fprintf(fid, '*SET,%%_FNCNAME%%(2,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(3,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(4,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(5,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(6,0,1), 0.0\n');
% Statement of the TIME variable. Stored in the first temporary var.
fprintf(fid, '*SET,%%_FNCNAME%%(0,1,1), 1.0, 99, 0, 1, 1, 0, 0\n');
% The rest of the operations to perform the equation have tha value 0
because any operation is needed to build the "TIME" equation.
fprintf(fid, '*SET,%%_FNCNAME%%(0,2,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,3,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,4,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,5,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,6,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,7,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,8,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,9,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,10,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,11,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,12,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,13,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,14,1),   0\n');
```

```
fprintf(fid, '*SET,%%_FNCNAME%%(0,15,1),    0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,16,1),    0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,17,1),    0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,18,1),    0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,19,1),    0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,20,1),    0\n');
% End of the TIME variable statement.
fprintf(fid, '! End of equation: {TIME}\n');
fprintf(fid, '!\n');
% Start of the REGIME 1 (Excitation). Signal equation.
fprintf(fid, ['! Begin of the equation: (0.5*(1-
cos((2*3.14*{TIME})/',num2str(T1),')))*sin(2*3.14*',num2str(F),'*\n']);
fprintf(fid, '! {TIME})\n');
% Statement of the ending time of the Excitation Period. 0.0002sec.
fprintf(fid, ['*SET,%%_FNCNAME%%(0,0,2), ',num2str(T1),', -999\n']);
fprintf(fid, '*SET,%%_FNCNAME%%(2,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(3,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(4,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(5,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(6,0,2), 0.0\n');
% Var. 1 = 2
fprintf(fid, '*SET,%%_FNCNAME%%(0,1,2), 1.0, -1, 0, 2, 0, 0, 0\n');
% Var. 2 = 3.14
fprintf(fid, '*SET,%%_FNCNAME%%(0,2,2), 0.0, -2, 0, 3.14, 0, 0, -1\n');
% Var. 3 = Var. 1 * Var. 2 = 3*3.14 (Product operation is shown with the
number 3 (see Operations Table))
fprintf(fid, '*SET,%%_FNCNAME%%(0,3,2),   0, -3, 0, 1, -1, 3, -2\n');
% Var. 1 = Var.3 * TIME = 2*3.14*TIME (TIME is shown with the ending 1 (See
Variables Table))
fprintf(fid, '*SET,%%_FNCNAME%%(0,4,2), 0.0, -1, 0, 1, -3, 3, 1\n');
% Var. 2 = 0.0002
fprintf(fid, ['*SET,%%_FNCNAME%%(0,5,2), 0.0, -2, 0, ',num2str(T1),', 0, 0,
-1\n']);
% Var. 3 = Var. 1 / Var. 2 = (2*3.14*TIME)/0.0002 (Division is shown with
the number 4 (See operations table))
fprintf(fid, '*SET,%%_FNCNAME%%(0,6,2), 0.0, -3, 0, 1, -1, 4, -2\n');
% Var. 1 = cos (Var.3 ) = cos ((2*3.14*TIME)/0.0002) (cosinus is shown with
the operation number 10).
fprintf(fid, '*SET,%%_FNCNAME%%(0,7,2), 0.0, -1, 10, 1, -3, 0, 0\n');
% Var. 2 = 1
fprintf(fid, '*SET,%%_FNCNAME%%(0,8,2), 0.0, -2, 0, 1, 0, 0, -1\n');
% Var. 3 = Var. 2 - Var. 1 = 1-cos ((2*3.14*TIME)/0.0002) (Substract
operation is show with the number 2 (See operations table))
fprintf(fid, '*SET,%%_FNCNAME%%(0,9,2), 0.0, -3, 0, 1, -2, 2, -1\n');
% Var. 1 = 0.5
fprintf(fid, '*SET,%%_FNCNAME%%(0,10,2), 0.0, -1, 0, 0.5, 0, 0, -3\n');
% Var. 2 = Var. 1 = Var. 3 = 0.5*(1-cos ((2*3.14*TIME)/0.0002))
fprintf(fid, '*SET,%%_FNCNAME%%(0,11,2), 0.0, -2, 0, 1, -1, 3, -3\n');
% Var. 1 = 2
fprintf(fid, '*SET,%%_FNCNAME%%(0,12,2), 0.0, -1, 0, 2, 0, 0, 0\n');
% Var. 3 = 3.14
fprintf(fid, '*SET,%%_FNCNAME%%(0,13,2), 0.0, -3, 0, 3.14, 0, 0, -1\n');
% Var. 4 = Var. 1 * Var. 3 = 2*3.14
fprintf(fid, '*SET,%%_FNCNAME%%(0,14,2), 0.0, -4, 0, 1, -1, 3, -3\n');
% Var. 1 = Frequency = 50000
fprintf(fid, ['*SET,%%_FNCNAME%%(0,15,2), 0.0, -1, 0, ',num2str(F),', 0, 0,
-4\n']);
% Var. 3 = Var. 4 * Var. 1 = 2*3.14*50000
fprintf(fid, '*SET,%%_FNCNAME%%(0,16,2), 0.0, -3, 0, 1, -4, 3, -1\n');
% Var. 1 = Var. 3 * TIME = 2*3.14*50000*TIME
fprintf(fid, '*SET,%%_FNCNAME%%(0,17,2), 0.0, -1, 0, 1, -3, 3, 1\n');
% Var. 1 = sin (Var. 1) = sin (2*3.14*50000*TIME) (sinus operation is shown
with the number 9 (see table of operations))
fprintf(fid, '*SET,%%_FNCNAME%%(0,18,2), 0.0, -1, 9, 1, -1, 0, 0\n');
% Var. 3 = Var. 2 * Var. 1 = 0.5*(1-cos ((2*3.14*TIME)/0.0002)) * sin
(2*3.14*50000*TIME)
fprintf(fid, '*SET,%%_FNCNAME%%(0,19,2), 0.0, -3, 0, 1, -2, 3, -1\n');
% Final of equation
fprintf(fid, '*SET,%%_FNCNAME%%(0,20,2), 0.0, 99, 0, 1, -3, 0, 0\n');
% Ending of statement of the Regime 1.
```

```
fprintf(fid, ['! END of the equation: (0.5*(1-
cos((2*3.14*{TIME})/',num2str(T1),')))*sin(2*3.14*',num2str(F),'*\n']);
fprintf(fid, '! {TIME})\n');
fprintf(fid, '!\n');
% Start of the Regime 2 (Propagation)
fprintf(fid, '! Begin of equation: 0*{TIME}\n');
% Statement of the ending Regime 1 time = 0.001s.
fprintf(fid, ['*SET,%%_FNCNAME%%(0,0,3), ',num2str(T),', -999\n']);
fprintf(fid, '*SET,%%_FNCNAME%%(2,0,3), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(3,0,3), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(4,0,3), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(5,0,3), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(6,0,3), 0.0\n');
% Var. 1 = 0
fprintf(fid, '*SET,%%_FNCNAME%%(0,1,3), 1.0, -1, 0, 0, 0, 0, 1\n');
% var. 2 = Var. 1 * TIME = 0*TIME
fprintf(fid, '*SET,%%_FNCNAME%%(0,2,3), 0.0, -2, 0, 1, -1, 3, 1\n');
% Ending of the Regime 2 equation
fprintf(fid, '*SET,%%_FNCNAME%%(0,3,3),   0, 99, 0, 1, -2, 0, 0\n');
% % The rest of the operations to perform the equation have tha value 0
% because any operation is needed to build the "0*TIME" equation.
fprintf(fid, '*SET,%%_FNCNAME%%(0,4,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,5,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,6,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,7,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,8,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,9,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,10,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,11,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,12,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,13,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,14,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,15,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,16,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,17,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,18,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,19,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,20,3),   0\n');
fprintf(fid, '! End of equation: {TIME}\n');
fprintf(fid, '!\n');

% Force on node

fprintf(fid, '!*\n');
fprintf(fid, 'FLST,2,1,1,ORDE,1\n');
% Application of the force in the middle node from the left edge of plate
fprintf(fid, ['FITEM,2,',num2str(10001+(10000*(Nt/2))),'\n']);
fprintf(fid, '/GO\n');
fprintf(fid, '!*\n');
fprintf(fid, '!*\n');
% Application of a Force, in the Y-axis direction. HANNING table values.
fprintf(fid, 'F,P51X,FY,%%HANNING%%\n');
fprintf(fid, '!*\n');

    case 'hamming'

fprintf(fid, '*DEL,_FNCNAME\n');
fprintf(fid, '*DEL,_FNCMTID\n');
fprintf(fid, '*DEL,_FNCCSYS\n');
% Name of the table where the values are stored
fprintf(fid, '*SET,_FNCNAME,''HAMMING''\n');
fprintf(fid, '*SET,_FNCCSYS,0\n');
% Import data from the .func file corresponding to the HAMMING window
fprintf(fid, '! /INPUT,.\\functions\\hamm.func,,,1\n');
% Table Values. 20 operations to compose the equation. 3 regimes
fprintf(fid, '*DIM,%%_FNCNAME%%,TABLE,6,20,3,,,,%%_FNCCSYS%%\n');
fprintf(fid, '!\n');
% Statement of the variable TIME
fprintf(fid, '! Begin of equation: {TIME}\n');
% -999 represents function in table format
```

```
fprintf(fid, '*SET,%%_FNCNAME%%(0,0,1), 0.0, -999\n');
fprintf(fid, '*SET,%%_FNCNAME%%(2,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(3,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(4,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(5,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(6,0,1), 0.0\n');
% Statement of the TIME variable. Stored in the first temporary var.
fprintf(fid, '*SET,%%_FNCNAME%%(0,1,1), 1.0, 99, 0, 1, 1, 0, 0\n');
% The rest of the operations to perform the equation have tha value 0
because any operation is needed to build the "TIME" equation.
fprintf(fid, '*SET,%%_FNCNAME%%(0,2,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,3,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,4,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,5,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,6,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,7,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,8,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,9,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,10,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,11,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,12,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,13,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,14,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,15,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,16,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,17,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,18,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,19,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,20,1),   0\n');
% End of the TIME variable statement
fprintf(fid, '! End of equation: {TIME}\n');
fprintf(fid, '!\n');
% Start of the REGIME 1 (Excitation). Signal equation.
fprintf(fid, ['! Begin of the equation: (0.53836-
(0.46164*cos(2*3.14*{TIME})/',num2str(T1),')))*sin(2*3.14*',num2str(F),'*\n'
]);
fprintf(fid, '! {TIME})\n');
% Statement of the ending time of the Excitation Period. 0.0002sec
fprintf(fid, ['*SET,%%_FNCNAME%%(0,0,2), ',num2str(T1),', -999\n']);
fprintf(fid, '*SET,%%_FNCNAME%%(2,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(3,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(4,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(5,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(6,0,2), 0.0\n');
% Var. 1 = 2
fprintf(fid, '*SET,%%_FNCNAME%%(0,1,2), 1.0, -1, 0, 2, 0, 0, 0\n');
% Var. 2 = 3.14
fprintf(fid, '*SET,%%_FNCNAME%%(0,2,2), 0.0, -2, 0, 3.14, 0, 0, -1\n');
% Var. 3 = Var. 1 * Var. 2 = 2*3.14
fprintf(fid, '*SET,%%_FNCNAME%%(0,3,2),   0, -3, 0, 1, -1, 3, -2\n');
% Var. 1 = Var. 3 * TIME = 2*3.14*TIME
fprintf(fid, '*SET,%%_FNCNAME%%(0,4,2), 0.0, -1, 0, 1, -3, 3, 1\n');
% Var 2 = 0.0002
fprintf(fid, ['*SET,%%_FNCNAME%%(0,5,2), 0.0, -2, 0, ',num2str(T1),', 0, 0,
-1\n']);
% Var. 3 = Var. 1 / Var. 2 = (2*3.14*TIME)/0.0002
fprintf(fid, '*SET,%%_FNCNAME%%(0,6,2), 0.0, -3, 0, 1, -1, 4, -2\n');
% Var. 1 = cos (Var. 3) = cos ((2*3.14*TIME)/0.0002)
fprintf(fid, '*SET,%%_FNCNAME%%(0,7,2), 0.0, -1, 10, 1, -3, 0, 0\n');
% Var. 2 = 0.46164
fprintf(fid, '*SET,%%_FNCNAME%%(0,8,2), 0.0, -2, 0, 0.46164, 0, 0, -1\n');
% Var. 3 = Var. 2 * Var. 1 = 0.46164 * cos ((2*3.14*TIME)/0.0002)
fprintf(fid, '*SET,%%_FNCNAME%%(0,9,2), 0.0, -3, 0, 1, -2, 3, -1\n');
% Var. 1 = 0.53836
fprintf(fid, '*SET,%%_FNCNAME%%(0,10,2), 0.0, -1, 0, 0.53836, 0, 0, -3\n');
% Var. 2 = Var. 1 - Var. 3 = 0.53836 - 0.46164 * cos ((2*3.14*TIME)/0.0002)
fprintf(fid, '*SET,%%_FNCNAME%%(0,11,2), 0.0, -2, 0, 1, -1, 2, -3\n');
% Var. 1 = 2
fprintf(fid, '*SET,%%_FNCNAME%%(0,12,2), 0.0, -1, 0, 2, 0, 0, 0\n');
% Var. 3 = 3.14
```

```
fprintf(fid, '*SET,%%_FNCNAME%%(0,13,2), 0.0, -3, 0, 3.14, 0, 0, -1\n');
% Var. 4 = Var. 1 * Var. 3 = 2*3.14
fprintf(fid, '*SET,%%_FNCNAME%%(0,14,2), 0.0, -4, 0, 1, -1, 3, -3\n');
% Var. 1 = Frequency = 50000
fprintf(fid, ['*SET,%%_FNCNAME%%(0,15,2), 0.0, -1, 0, ',num2str(F),', 0, 0,
-4\n']);
% Var. 3 = Var. 4 * Var. 1 = 2*3.14*50000
fprintf(fid, '*SET,%%_FNCNAME%%(0,16,2), 0.0, -3, 0, 1, -4, 3, -1\n');
% Var. 1 = Var. 3 * TIME = 2*3.14*50000*TIME
fprintf(fid, '*SET,%%_FNCNAME%%(0,17,2), 0.0, -1, 0, 1, -3, 3, 1\n');
% Var. 1 = sin(Var. 1) = sin(2*3.14*50000*TIME)
fprintf(fid, '*SET,%%_FNCNAME%%(0,18,2), 0.0, -1, 9, 1, -1, 0, 0\n');
% Var. 3 = Var. 2 * Var. 1 = (0.53836 - 0.46164 *
cos((2*3.14*TIME)/0.0002))*sin(2*3.14*50000*TIME)
fprintf(fid, '*SET,%%_FNCNAME%%(0,19,2), 0.0, -3, 0, 1, -2, 3, -1\n');
% Final of equation
fprintf(fid, '*SET,%%_FNCNAME%%(0,20,2), 0.0, 99, 0, 1, -3, 0, 0\n');
fprintf(fid, ['! End of the equation: (0.54-
(0.46*cos(2*3.14*{TIME})/',num2str(T1),')))*sin(2*3.14*',num2str(F),'*\n']);
fprintf(fid, '! {TIME})\n');
fprintf(fid, '!\n');
% Start of the Regime 2 (Propagation)
fprintf(fid, '! Begin of equation: 0*{TIME}\n');
% Statement of the ending Regime 1 time = 0.001s.
fprintf(fid, ['*SET,%%_FNCNAME%%(0,0,3), ',num2str(T),', -999\n']);
fprintf(fid, '*SET,%%_FNCNAME%%(2,0,3), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(3,0,3), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(4,0,3), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(5,0,3), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(6,0,3), 0.0\n');
% Var. 1 = 0
fprintf(fid, '*SET,%%_FNCNAME%%(0,1,3), 1.0, -1, 0, 0, 0, 0, 1\n');
% Var. 2 = Var. 1 * TIME = 0*TIME
fprintf(fid, '*SET,%%_FNCNAME%%(0,2,3), 0.0, -2, 0, 1, -1, 3, 1\n');
% End of equation
fprintf(fid, '*SET,%%_FNCNAME%%(0,3,3),   0, 99, 0, 1, -2, 0, 0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,4,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,5,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,6,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,7,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,8,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,9,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,10,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,11,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,12,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,13,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,14,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,15,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,16,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,17,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,18,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,19,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,20,3),   0\n');
fprintf(fid, '! End of equation: {TIME}\n');
fprintf(fid, '!\n');

% Force on node

fprintf(fid, '!*\n');
fprintf(fid, 'FLST,2,1,1,ORDE,1\n');
% Application of the force in the middle node from the left edge of plate
fprintf(fid, ['FITEM,2,',num2str(10001+(10000*(Nt/2))),'\n']);
fprintf(fid, '/GO\n');
fprintf(fid, '!*\n');
fprintf(fid, '!*\n');
% Application of a Force, in the Y-axis direction. HAMMING table values.
fprintf(fid, 'F,P51X,FY,%%HAMMING%%\n');
fprintf(fid, '!*\n');
```

```
    case 'gauss'

fprintf(fid, '*DEL,_FNCNAME\n');
fprintf(fid, '*DEL,_FNCMTID\n');
fprintf(fid, '*DEL,_FNCCSYS\n');
% Name of the table where all the function values are stored.
fprintf(fid, '*SET,_FNCNAME,''GAUSS''\n');
fprintf(fid, '*SET,_FNCCSYS,0\n');
% Importation data from the .func file corresponding to the Gauss Function.
fprintf(fid, '! /INPUT,.\\functions\\gauss.func,,,1\n');
% Gauss table: 28 operations. 3 Regimes.
fprintf(fid, '*DIM,%%_FNCNAME%%,TABLE,6,28,3,,,,%%_FNCCSYS%%\n');
fprintf(fid, '!\n');
% Statement of the TIME variable
fprintf(fid, '! Begin of equation: {TIME}\n');
% -999 represents function in table format
fprintf(fid, '*SET,%%_FNCNAME%%(0,0,1), 0.0, -999\n');
fprintf(fid, '*SET,%%_FNCNAME%%(2,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(3,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(4,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(5,0,1), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(6,0,1), 0.0\n');
%  Statement of the TIME variable.
fprintf(fid, '*SET,%%_FNCNAME%%(0,1,1), 1.0, 99, 0, 1, 1, 0, 0\n');
% The rest of the operations to perform the equation have tha value 0
because any operation is needed to build the "TIME" equation.
fprintf(fid, '*SET,%%_FNCNAME%%(0,2,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,3,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,4,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,5,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,6,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,7,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,8,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,9,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,10,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,11,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,12,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,13,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,14,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,15,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,16,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,17,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,18,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,19,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,20,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,21,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,22,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,23,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,24,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,25,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,26,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,27,1),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,28,1),   0\n');
fprintf(fid, '! End of equation: {TIME}\n');
fprintf(fid, '!\n');
% Start of the REGIME 1 (Excitation). Signal equation.
fprintf(fid, ['! Begin of the equation: exp(-0.5*((({TIME}-
(',num2str(T1),'/2))/(0.275*',num2str(T1),'/2))^2))*sin(2\n']);
fprintf(fid, ['! *3.14*',num2str(F),'*{TIME})\n']);
% Statement of the ending propagation time = 0.0002sec
fprintf(fid, ['*SET,%%_FNCNAME%%(0,0,2), ',num2str(T1),', -999\n']);
fprintf(fid, '*SET,%%_FNCNAME%%(2,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(3,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(4,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(5,0,2), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(6,0,2), 0.0\n');
% Var. 1 = 0
fprintf(fid, '*SET,%%_FNCNAME%%(0,1,2), 1.0, -1, 0, 0, 0, 0, 0\n');
% Var. 2 = 1
```

```
fprintf(fid, '*SET,%%_FNCNAME%%(0,2,2), 0.0, -2, 0, 1, 0, 0, -1\n');
% Var. 3 = Var. 1 - Var. 2 = -1
fprintf(fid, '*SET,%%_FNCNAME%%(0,3,2),   0, -3, 0, 1, -1, 2, -2\n');
% Var. 1 = 0.5
fprintf(fid, '*SET,%%_FNCNAME%%(0,4,2), 0.0, -1, 0, 0.5, 0, 0, -3\n');
% Var. 2 = Var. 3 * var. 1 = -0.5
fprintf(fid, '*SET,%%_FNCNAME%%(0,5,2), 0.0, -2, 0, 1, -3, 3, -1\n');
% Var. 1 = 0.0002
fprintf(fid, ['*SET,%%_FNCNAME%%(0,6,2), 0.0, -1, 0, ',num2str(T1),', 0, 0,
0\n']);
% Var. 3 = 2
fprintf(fid, '*SET,%%_FNCNAME%%(0,7,2), 0.0, -3, 0, 2, 0, 0, -1\n');
% Var. 4 = Var. 1 / Var. 3 = 0.0002/2
fprintf(fid, '*SET,%%_FNCNAME%%(0,8,2), 0.0, -4, 0, 1, -1, 4, -3\n');
% Var. 1 = TIME - Var. 4 = TIME - (0.0002/2)
fprintf(fid, '*SET,%%_FNCNAME%%(0,9,2), 0.0, -1, 0, 1, 1, 2, -4\n');
% Var. 3 = 0.275
fprintf(fid, '*SET,%%_FNCNAME%%(0,10,2), 0.0, -3, 0, 0.275, 0, 0, 0\n');
% Var. 4 = 0.0002
fprintf(fid, ['*SET,%%_FNCNAME%%(0,11,2), 0.0, -4, 0, ',num2str(T1),', 0, 0,
-3\n']);
% Var. 5 = Var. 3 * Var. 4 = 0.275*0.0002
fprintf(fid, '*SET,%%_FNCNAME%%(0,12,2), 0.0, -5, 0, 1, -3, 3, -4\n');
% Var. 3 = 2
fprintf(fid, '*SET,%%_FNCNAME%%(0,13,2), 0.0, -3, 0, 2, 0, 0, -5\n');
% Var. 4 = Var. 5 / Var. 3 = 0.275*0.0002/2
fprintf(fid, '*SET,%%_FNCNAME%%(0,14,2), 0.0, -4, 0, 1, -5, 4, -3\n');
% Var. 3 = Var. 1 / Var. 4 = (TIME-(0.0002/2))/(0.275*0.0002/2)
fprintf(fid, '*SET,%%_FNCNAME%%(0,15,2), 0.0, -3, 0, 1, -1, 4, -4\n');
% Var. 1 = 2
fprintf(fid, '*SET,%%_FNCNAME%%(0,16,2), 0.0, -1, 0, 2, 0, 0, -3\n');
% Var. 4 = Var. 3 ^ Var. 1 = [(TIME-(0.0002/2))/(0.275*0.0002/2)]^2 (power
operation is shown with the number 17 in the operations table)
fprintf(fid, '*SET,%%_FNCNAME%%(0,17,2), 0.0, -4, 0, 1, -3, 17, -1\n');
% Var. 1 = Var. 2 * Var. 4 = -0.5*[[(TIME-(0.0002/2))/(0.275*0.0002/2)]^2]
fprintf(fid, '*SET,%%_FNCNAME%%(0,18,2), 0.0, -1, 0, 1, -2, 3, -4\n');
% Var. 1 = e^(-0.5*[[(TIME-(0.0002/2))/(0.275*0.0002/2)]^2]) (power e
operation is shown with the number 7 in the operations table.)
fprintf(fid, '*SET,%%_FNCNAME%%(0,19,2), 0.0, -1, 7, 1, -1, 0, 0\n');
% Var. 2 = 2
fprintf(fid, '*SET,%%_FNCNAME%%(0,20,2), 0.0, -2, 0, 2, 0, 0, 0\n');
% Var. 3 = 3.14
fprintf(fid, '*SET,%%_FNCNAME%%(0,21,2), 0.0, -3, 0, 3.14, 0, 0, -2\n');
% Var. 4 = Var. 2 * Var. 3 =2*3.14
fprintf(fid, '*SET,%%_FNCNAME%%(0,22,2), 0.0, -4, 0, 1, -2, 3, -3\n');
% Var. 2 = Frequency = 50000
fprintf(fid, ['*SET,%%_FNCNAME%%(0,23,2), 0.0, -2, 0, ',num2str(F),',1 0, 0,
-4\n']);
% Var. 3 = Var. 4 * Var. 2 = 2*3.14*50000
fprintf(fid, '*SET,%%_FNCNAME%%(0,24,2), 0.0, -3, 0, 1, -4, 3, -2\n');
% Var. 2 = Var. 3 * TIME = 2*3.14*50000*TIME
fprintf(fid, '*SET,%%_FNCNAME%%(0,25,2), 0.0, -2, 0, 1, -3, 3, 1\n');
% Var. 2 = sin(Var. 2) = sin (2*3.14*50000*TIME)
fprintf(fid, '*SET,%%_FNCNAME%%(0,26,2), 0.0, -2, 9, 1, -2, 0, 0\n');
% Var. 3 = Var. 1 * Var. 2 = e^(-0.5*[[(TIME-
(0.0002/2))/(0.275*0.0002/2)]^2])*sin (2*3.14*50000*TIME)
fprintf(fid, '*SET,%%_FNCNAME%%(0,27,2), 0.0, -3, 0, 1, -1, 3, -2\n');
% End of the equation
fprintf(fid, '*SET,%%_FNCNAME%%(0,28,2), 0.0, 99, 0, 1, -3, 0, 0\n');
fprintf(fid, ['! End of the equation: exp(-0.5*((({TIME}-
(',num2str(T1),'/2))/(0.275*',num2str(T1),'/2))^2))*sin(2\n']);
fprintf(fid, ['! *3.14*',num2str(F),'*{TIME})\n']);
fprintf(fid, '!\n');
% Start of the REGIME 2. (Propagation)
fprintf(fid, '! Begin of equation: 0*{TIME}\n');
% Statement of the ending propagation time = 0.001sec
fprintf(fid, ['*SET,%%_FNCNAME%%(0,0,3), ',num2str(T),', -999\n']);
fprintf(fid, '*SET,%%_FNCNAME%%(2,0,3), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(3,0,3), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(4,0,3), 0.0\n');
```

```
fprintf(fid, '*SET,%%_FNCNAME%%(5,0,3), 0.0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(6,0,3), 0.0\n');
% Var. 1 = 0
fprintf(fid, '*SET,%%_FNCNAME%%(0,1,3), 1.0, -1, 0, 0, 0, 0, 1\n');
% Var. 2 = Var. 1 * TIME = 0*TIME
fprintf(fid, '*SET,%%_FNCNAME%%(0,2,3), 0.0, -2, 0, 1, -1, 3, 1\n');
% End of the equation
fprintf(fid, '*SET,%%_FNCNAME%%(0,3,3),   0, 99, 0, 1, -2, 0, 0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,4,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,5,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,6,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,7,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,8,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,9,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,10,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,11,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,12,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,13,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,14,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,15,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,16,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,17,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,18,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,19,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,20,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,21,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,22,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,23,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,24,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,25,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,26,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,27,3),   0\n');
fprintf(fid, '*SET,%%_FNCNAME%%(0,28,3),   0\n');
fprintf(fid, '! End of equation: {TIME}\n');
fprintf(fid, '!\n');

% Force on node

fprintf(fid, '!*\n');
fprintf(fid, 'FLST,2,1,1,ORDE,1\n');
% Application of the force in the middle node from the left edge of plat
fprintf(fid, ['FITEM,2,',num2str(10001+(10000*(Nt/2))),'\n']);
fprintf(fid, '/GO\n');
fprintf(fid, '!*\n');
fprintf(fid, '!*\n');
% Application of a Force, in the Y-axis direction. GAUSS table values.
fprintf(fid, 'F,P51X,FY,%%GAUSS%%\n');
fprintf(fid, '!*\n');

end
```

4.8 Initial Conditions.

With the purpose of enforcing the coded signals in their corresponding node, the command “FSLT”[2], whose input data structure was “**FLST**, NFIELD, NARG, TYPE, Otype, LENG”, was used. In this input data structure, the “NFIELD” term was the field number on the command which used the picking data, “NARG” the number of items

[2] This command served for specifying the data required for a picking operation.

in the picked list, “TYPE” was the type if the elements picked[3], “Otype” indicated if the data was ordered or not and finally “LENG” specified the length of number of items describing the list. In this case, “NFIELD” was equal to 2 (because it indicated the first command argument), “NARG” was equal to 1 due to the fact that just one node was picked, “TYPE” was number 1 as we were working with nodes, “Otype” was “ORDE” as the data was in order and finally “LENG” was equal to 1 because we worked with just one node.

Furthermore, it was necessary to use the “FITEM” command, whose utility was to identify the item chosen by a picking operation; its input data structure was “**FITEM**, NFIELD, ITEM, ITEMY, ITEMZ”, where “NFIELD” was the same as in the “FLST” command and “ITEM” was the id of the picked item.

Finally, in order to apply the force to the picked node, it was required to employ the “F” command, whose input data was composed by the following terms: “NODE” (node at which the force was specified)[4], “Lab” (Valid Force Label: FX/FY/FZ…), “VALUE” (indicated the constant value of the force or refers to an existing table), “VALUE2” (the second value, if any), “NEND” and “NINC”, that were useless for our instruction.

After completing this stage of the simulation, what missed was setting the Initial Conditions; to this end, the command “IC” was employed. Both the “NODE” and the “VALUE” terms were “ALL” as we wanted to picked all the selected nodes and use as well all the appropriate labels.

```
% SET INITIAL CONDITIONS

fprintf(fid, 'IC,ALL,ALL\n');
fprintf(fid, '!*\n');
```

4.9 Timing and frequency settings.

The last phase to finish the script and then running on the program was stating the time and frequency parameters such as the total simulation time or the time step.

```
%TIME-TIME SETTINGS

fprintf(fid, 'TIME,0.001\n');
fprintf(fid, 'AUTOTS,0\n');
```

[3] 1 = Nodes; 2 = Elements; 3 = Keypoints; 4 = Lines; 5 = Areas; 6 = Volumes.

[4] In our case, “NODE” was “P51X” because it referred to the previously picked node.

```
fprintf(fid,
['DELTIM,',num2str(0.5*dt),',',num2str(0.1*dt),',',num2str(dt),',1\n']);
fprintf(fid, 'KBC,1\n');
fprintf(fid, '!*\n');
fprintf(fid, 'TSRES,ERASE\n');
fprintf(fid, '!*\n');
```

The reason why the command “TIME” was used was because of the need to define the total time simulation (also called load step). As it was mentioned previously, the simulation would take 0.001 s in order to let the signal be propagated though the end of the plate but avoiding at the same time the existence of rebounds. Then, it was required to chose whether to use the automatic time stepping or the load stepping. Here again ANSYS worked with the binary choice system stated in the previous simulation; the number 0 accompanied the command due to the simulation worked better without the automatic time stepping. Furthermore, disabling it allowed having control over the simulation.

It was required too to specify the time step sizes for that load step. The command that would perform that task was “DELTIM”, whose input data structure was “**DELTIM**, DTIME, DTMIN, DTMAX, Carry”, where “DTIME” was the time step size for the step, “DTMIN” and “DTMAX” were the minimum and maximum time steps respectively (whose instructions were defined in the previous simulation) and finally, the term “Carry” was the possibility to able/disable the time step carry-over key. If we remember the instructions of the previous simulation, we would notice that the time step was one half of the time considered (0.5*1E-6 s), where the minimum and the maximum were respectively one tenth of the time and the whole time step needed. Finally, it was precise to specify whether to use a ramped or a stepped load step. As before, it was needed to be stepped in order to load the whole force from the beginning; that was why the number 1 (referring to the second option) accompanied the command.

The last stage before running the script and obtaining the ANSYS input data file was controlling the solution data written to the database; the way to achieve it was by using the command “OUTRES”. This meant which of the data results would be able to be checked once the simulation was finished. As it was desirable to have all kind of results, the option “ALL” was selected. Then, it was required to specify the output frequency; in other words, the number of substeps of each load step that would take ANSYS to write a solution. It was needed to have as much as results as possible to build accurately a displacements’ graph, that was why it was imposed to ANSYS to take a solution each 2 subteps.

```
% DB/RESULT FILE SETTINGS

fprintf(fid, 'OUTRES,ALL,2\n');
fprintf(fid, '!*\n');
```

Once the script was executed and the input file was generated, then it was needed to open **ANSYS Mechanical APDL Product Launcher 14.0**. After launching it, it was necessary to open the file menu and select the option **"Read input from…"** and then choose the .lgw file created in the corresponding directory. In order to solve the simulation, those steps needed to be followed.

Menu ➔ File ➔ Read Input from…
Main Menu ➔ Solution ➔ Solve ➔ Current LS.

When the simulation ended, the same analysis procedure was repeated in order to check that the simulation and the code were correct. The attained results were very similar to the ones obtained with the previous simulation; despite having the same shape, the only difference between them was that the displacements corresponding to the recent simulation were a little bit smaller in comparison with the first simulation performed. The reason of this difference was because of the presence of the beginning absorbent strip; upon exciting the plate in the egde between the end of the strip and the beginning of the plate, part of the signal was attenuated with the strip.
Disperse showed its results without any absorbent strip, so that it was impossible to compare them exactly. As the displacements hat a very similar shape, despite differing a little bit in their amplitude, the simulation could be considered as correct. That fact meant that it could be compared then with the ABAQUS simulation. The attained results would be shown in the next chapter.

Chapter 5

Comparison results

5.1 Benchmarking

Before starting the comparison between both software solutions and with the purpose of verifying the stability of the second software, the simulation with ABAQUS was executed, always maintaining the same simulation parameters as it was done with ANSYS. The script that would generate the ABAQUS input data file was obtained through the École de Téchnologie Supérieure, from another project that had been realized.

Once all the parameters were correctly declared, it was noted that everything worked correctly except the time-step. Thus, in order to warranty ABAQUS' stability, it was precise to change it from 1E-6 s to 0.5E-7 s. Furthermore, the elements size had to be reduced from 0.00125 m to 0.000625 m in both programs. Once these items were arranged, the simulation was executed and the benchmarking[5] could be performed. In this comparison procedure, we analysed the memory that took the simulation to be performed, the File I/O (File Input/Output)[6] and the time necessary to complete the simulation.

With the purpose of obtaining the most accurate results, we did both simulations in the same laptop so as to avoid possible differences because of the computers characteristics. Those were the results attained after doing both simulations.

	Memory (Mb)	File i/o (Mb/s)	Total Time (h)
ANSYS	112	42.9	5.24
ABAQUS	111	25.1	0.1834

Analyzing the table results, it is easy to identify serious differences between both software solutions. As foreseeable, it looked logical that both programs used the

[5] We understand by Benchmarking the process of comparing one's practice with other industries. In our case, comparing the performances from two of the most important software solutions: ANSYS and ABAQUS.

[6] The File I/O is the writing or the reading of a file in a pc's hard drive. More the File I/O increases, more the pc's resources are used, which is obviously worse.

same memory as they were working exactly with the same model and the same number of elements. What differed was not only the File I/O but also, and in a significant way, the time taken by both software solutions to perform the simulation.

This was the first time Matlab was used to generate a script with the purpose of simulating Lamb Waves with ANSYS. Part of the time difference between those two software solutions may be attributed to that rawness. However, this inexperience is not the only cause for this huge difference between them. Another reason for this disparity may be the solution solver, that the ABAQUS one seemed to be quicker. It must be taken into account that the finite elements method consists on solving differential equations in order to obtain approximate solutions for boundary value problems. Depending on how the software solved those equations, the time taken to perform the simulation could be shorter or longer.

5.2 Displacement's results

After doing the benchmarking, what was done was comparing precisely the displacements' graphs offered by both programs; the excitation signal was always the same so what was interesting was analysing the node placed one meter far from the excitation node, in the middle of the thickness of the plate. Those were the displacements attained by ABAQUS after completing the simulation. To the naked eye, the graph looked very similar to the ones attained before, but once again it was required to obtain the y-axis displacement with ANSYS and compare both graphs with Matlab.

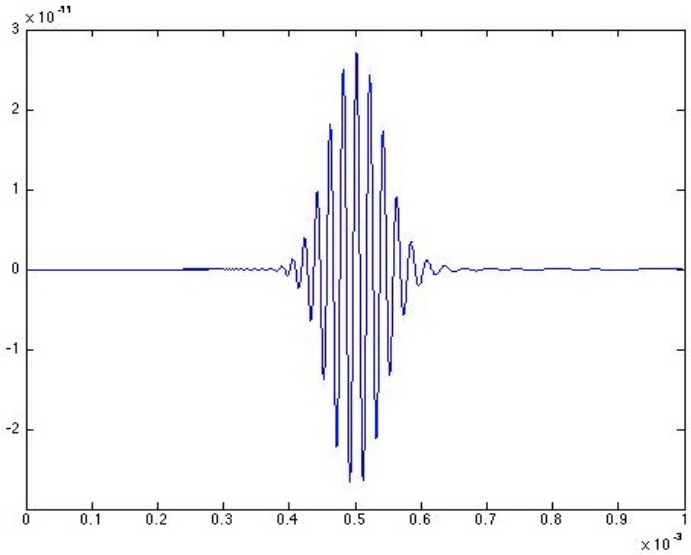


Figure 5.2.1: Y-axis displacement. Final simulation ABAQUS

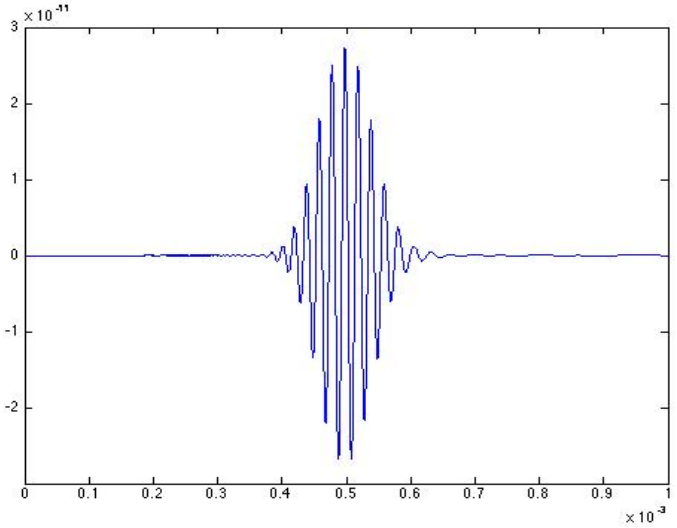


Figure 5.2.2: Y-axis displacement. Final simulation ANSYS

Both graphs looked really similar but these were the conclusions attained when they were superimposed.

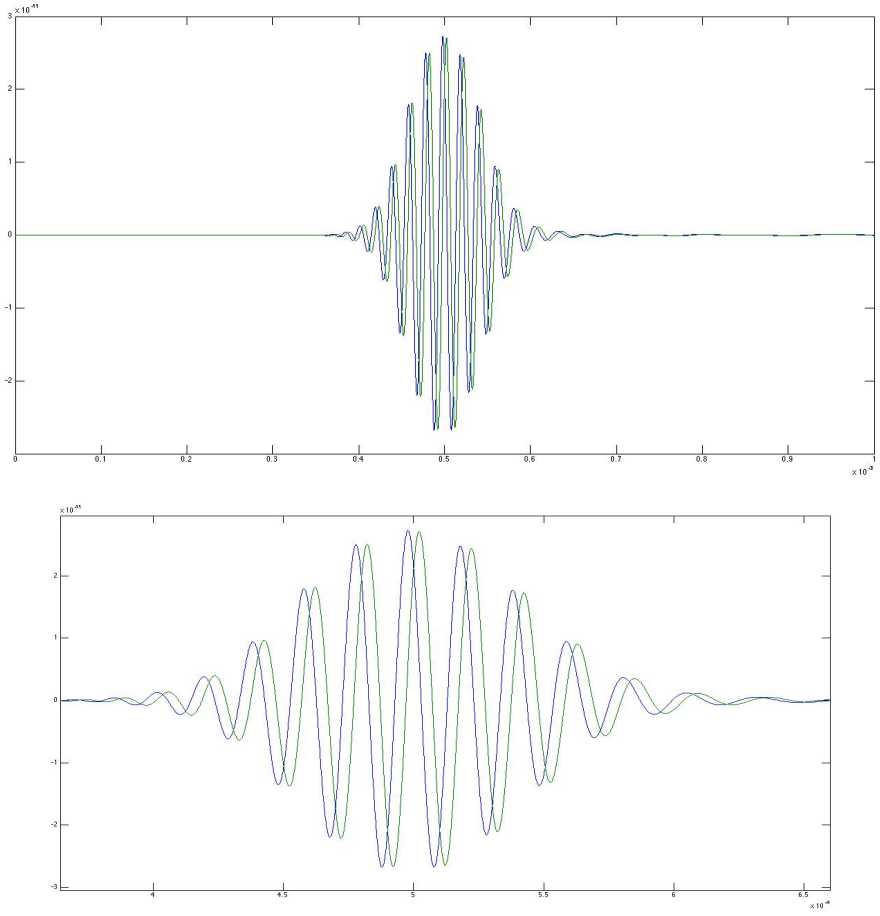


Figure 5.2.3: Comparison between ANSYS and ABAQUS

As it can be seen, both graphs looked identical. The only difference between them was that the displacements imported from ANSYS were displaced aprox 4.3E-6 s from the ones from ABAQUS. What this fact meant was that maybe the signal implemented in ANSYS began to be propagated some time after than the signal enforced in ABAQUS. Nevertheless, the error percent between both displacements' graphs was 0.87%, what signified that both of them could be considered correct.
With this comparison, the displacements' precision was discarded as a reason of choice between both software solutions given that both were equally exact. After this study then, it can be concluded then ABAQUS is a most powerful finite elements' method program as its resolution time and its File I/O is smaller than in ANSYS. Furthermore, the visual interface of this last program looks older and less good-looking, which tempts the user to choose ABAQUS before ANSYS.

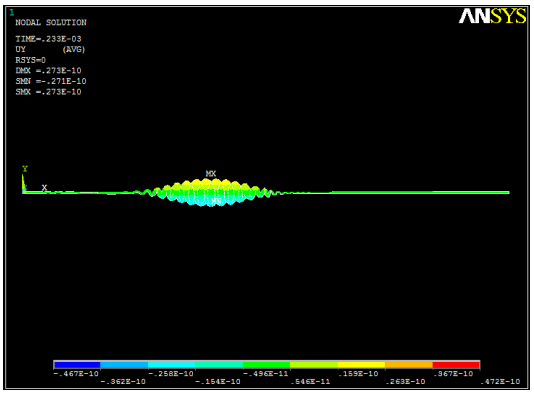
5.3 Pipelines' thickness study

Remembering what was mentioned in the abstract section in the beginning of the report about the pipeline's thickness, it was thought to present the results of a pipeline whose thickness had been damaged in order to see the difference between a spoiled conduit and a new one; this simulation showed how to appreciate if a stretch of a pipeline needed or not to be replaced.
The plate's dimensions were identical to the ones used in the previous simulations but this time a stretch of 0.2 m, whose thickness had been reduced to 2.5 mm was placed 0.5 m far from where the signal was enforced, was designed. Needless to say that just the initial parameters of the script had to be changed in order to perform theses changes.

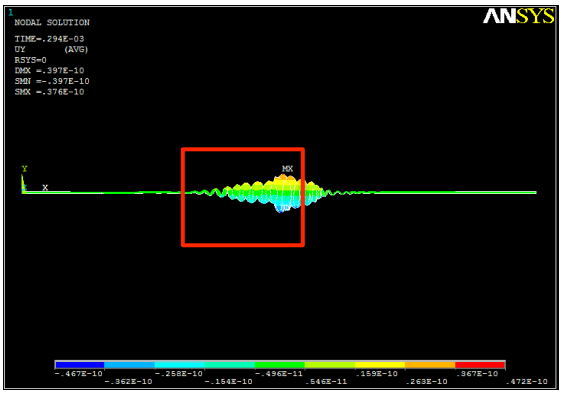
```
%% Default Geometry

% Diameter of the circular default place in the plate's surface
Defect_Diameter = [0.2];
% Depth of the circular default place in the plate's surface
Defect_Depth = [0.0025];
% Position where the plate defaut begins.
Defect_Location = [L/2];
```

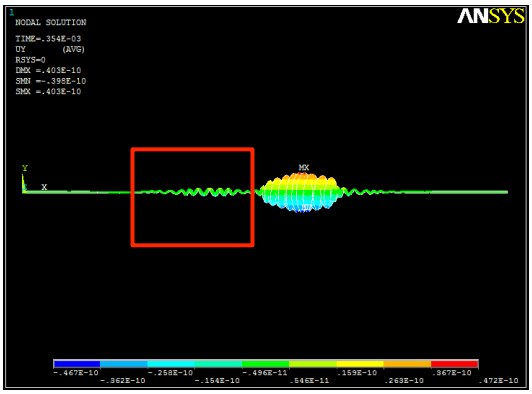
With the purpose of avoiding possible differences, the same node (placed 1m far from where the signal was implemented) was studied. Those were the attained results.



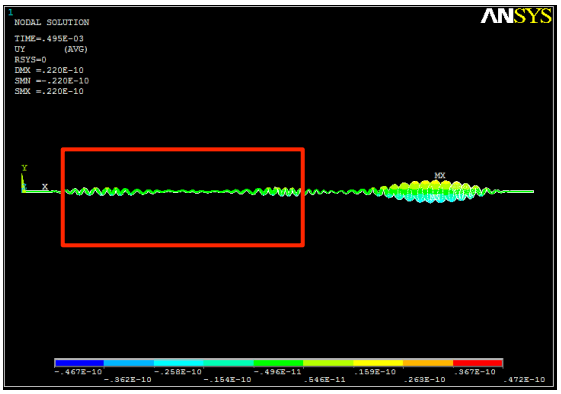
TIME = 0.233E-3 s



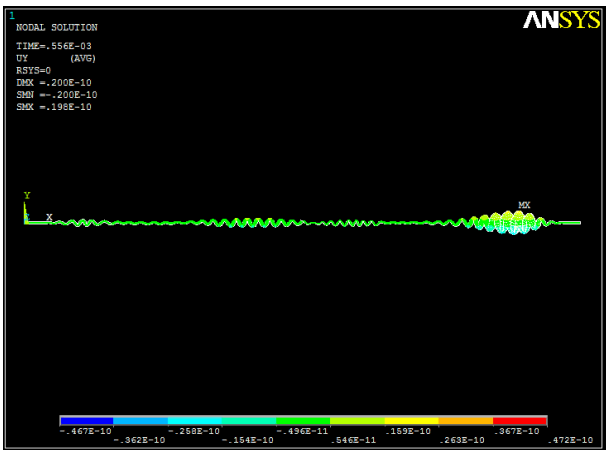
TIME = 0.294E-3 s



TIME = 0.354E-3 s



TIME = 0.495E-3 s



TIME = 0.556E-3s

Figure 5.3.1: ANSYS animation of the signal's propagation. Rebounds existence.

As it can be seen in the sequence of the animation's screenshots, a rebound was created as soon as the wave hit the flaw, in this case the stretch with the thickness damage. The wave was propagated correctly but when it reached the spoiled area, a wave with less amplitude was created. Finally both the signal and the rebound were attenuated by both of the absorbent strips.

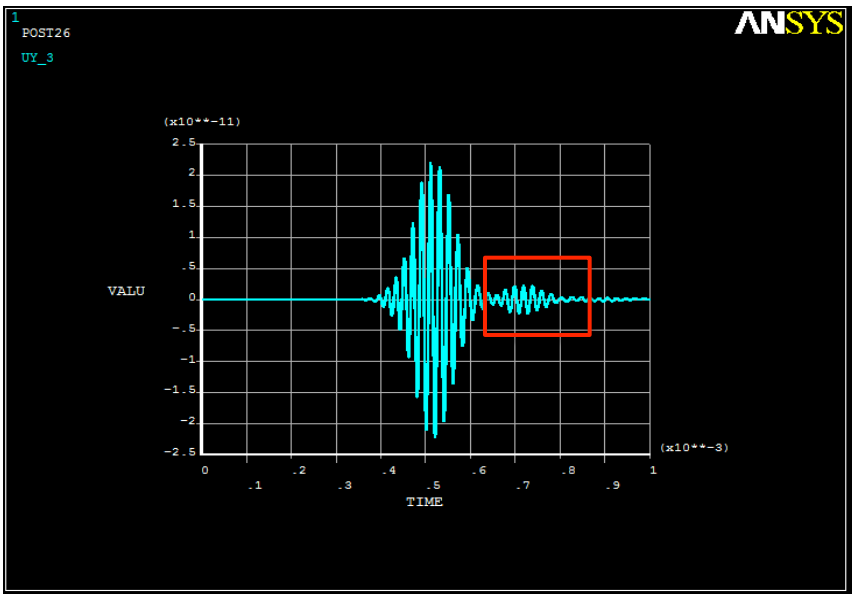


Figure 5.3.2: Displacement's graph for the studied node. Presence of a rebound.

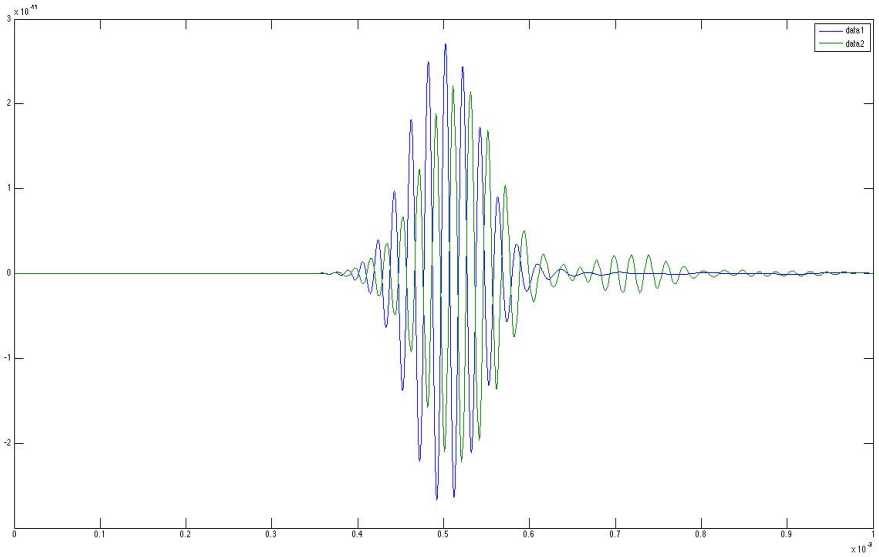


Figure 5.3.3: Comparison of the propagation of the wave in a damaged (red) and a new plate (blue).

Once the comparison was done, it maybe be realized that the way to identify a damaged stretch in a pipeline was by the appearance of a rebound in the displacement's graph besides an amplitude decrease.

In summary, this project has been very useful for learning about another usage of ANSYS, a software solution that may be very important in an engineer's professional life. Besides, all these tasks pushed to learn about how to code with Matlab, which was essential to perform the work. To all this acquired knowledge, it was given a utility by creating a little program with the aim of helping the FE simulations' users to choose the best software for their simulations.

References

PROPAGATION OF LAMB WAVES IN ANSYS

- Ramy Mohamed, Patrice Masson. *Modélisation de la propagation des ondes guidées*. SURVEILLANCE DE L'INTÉGRITÉ DES STRUCTURES - GMC 724, Juin 2011

ANSYS COMMANDS LIST

- http://mostreal.sk/html/c-index.htm
- ANSYS APDL Product Launcher 14.0 User's Help

ANSYS FUNCTION BUILDER

- https://engineering.purdue.edu/~abe601/ansys/transient_load_function_tutorial_v81.pdf
- http://ansys.net/ansys/tips/ANSYS_Function_Builder.pdf

LAMB WAVES THEORETICAL REVIEW

- J. David, N. Cheeke. *Fundamentals and Applications of Ultrasonic Waves.* CRC Press
- *P. Belanger. Feasibility of thickness mapping using ultrasonic waves.* Imperial College of London, November 2009.